IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 28 June 2022, accepted 4 July 2022, date of publication 25 July 2022, date of current version 28 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3190538

RESEARCH ARTICLE

Edge Deployment Framework of GuardBot for Optimized Face Mask Recognition With Real-Time Inference Using Deep Learning

SUMAIRA MANZOOR[1], EUN-JIN KIM[2], SUNG-HYEON JOO[2], SANG-HYEON BAE[2], GUN-GYO IN[2], KYEONG-JIN JOO[2], JUN-HYEON CHOI[2], AND TAE-YONG KUC[2]

[1]Creative Algorithms and Sensor Evolution Laboratory, Suwon 16419, South Korea
[2]Department of Electrical and Computer Engineering, College of Information and Communication Engineering, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Tae-Yong Kuc (tykuc@skku.edu)

This research is supported by the Korea Evaluation Institute of Industrial Technology (KEIT) funded by the Ministry of Trade, Industry, and Energy (MOTIE) (No. 1415180816).

ABSTRACT Deep learning based models on the edge devices have received considerable attention as a promising means to handle a variety of AI applications. However, deploying the deep learning models in the production environment with efficient inference on the edge devices is still a challenging task due to computation and memory constraints. This paper proposes a framework for the service robot named GuardBot powered by Jetson Xavier NX and presents a real-world case study of deploying the optimized face mask recognition application with real-time inference on the edge device. It assists the robot to detect whether people are wearing a mask to guard against COVID-19 and gives a polite voice reminder to wear the mask. Our framework contains dual-stage architecture based on convolutional neural networks with three main modules that employ (1) MTCNN for face detection, (2) our proposed CNN model and seven transfer learning based custom models which are Inception-v3, VGG16, denseNet121, resNet50, NASNetMobile, XceptionNet, MobileNet-v2 for face mask classification, (3) TensorRT for optimization of all the models to speedup inference on the Jetson Xavier NX. Our study carries out several analysis based on the models' performance in terms of their frames per second, execution time and images per second. It also evaluates the accuracy, precision, recall & F1-score and makes the comparison of all models before and after optimization with a main focus on high throughput and low latency. Finally, the framework is deployed on a mobile robot to perform experiments in both outdoor and multi-floor indoor environments with patrolling and non-patrolling modes. Compared to other state-of-the-art models, our proposed CNN model for face mask recognition based on the classification obtains 94.5%, 95.9% and 94.28% accuracy on training, validation and testing datasets respectively which is better than MobileNet-v2, Xception and InceptionNet-v3 while it achieves highest throughput and lowest latency than all other models after optimization at different precision levels.

INDEX TERMS Face detection, face mask classification, voice alert, service robot, deep learning, transfer learning, Jetson Xavier NX, TensorFlow TensorRT, optimization, inference.

ACRONYMS

The list of acronyms and abbreviations used in this survey is given in Table 1.

The associate editor coordinating the review of this manuscript and approving it for publication was Jamshid Aghaei.

I. INTRODUCTION

Deep learning [1] models based on CNN are key components for developing efficient robotic systems with a wide range of applications that span from vision-based recognition, NLP, UAV to entertainment and healthcare. The deployment strategies for these DL-based applications can be categorized into cloud-based and edge-based. Cloud-based deployment is

TABLE 1. List of Acronyms and Abbreviations used in this paper.

CNN	Convolutional Neural Network
NLP	Natural Language Processing,
UAV	Unmanned Aerial vehicles,
DL	Deep Learning
TRL	Transfer Learning
FCN	Fully Connected Network
ML	Machine Learning
IDC	International Data Corporation
SBC	Small Single-Board Computer
GPU	Graphics Processing Unit
ATR	dvanced Telecommunication Research
TRT	TensorRT
TFLite	TensorFlow Lite
Covid-19	Coronavirus 2019
NLP	Natural Language Processing
AWS	Amazon Web Services
AV	Autonomous Vehicles
HCW	Health Care Workers
HRI	Human Robot Interaction
FP64	Floating Point 64-bit
CDCP	Centers Disease Control and Prevention
SOPs	Standard Operating Procedures
WHO	Wold Health Organization
NCNN	Neural Network Inference Computing
SVM	Support Vector Machine
SMFD	Simulated Masked Face Dataset
LFW	Labeled Faces in the Wild
RMFD	Real-World Masked Face Dataset
MTCNN	Multi-task cascaded CNN
HOG	Histogram of Oriented Gradients
R-Net	Refine Network
O-Net	Output-Network)
P-Net	Proposal Network
DAG	Directed Acyclic Graphs
FC	Fully Connected
CUDA	Compute Unified Device Architecture
RNN	Recurrent neural networks
ASR	Automatic Speech Recognition
FPS	Frames Per Second

centralized with high processing and compute power, while edge-based deployment is decentralized with low-latency [2]. Despite the better performance of cloud-based AI solutions, most DL-based applications cannot compromise with the high latency rate in data transfer and security threats in the network. Hence, edge-based deployment is an alternate solution for building faster, more secure and easily scalable applications [3]. More recent, deep learning models' deployment at the edge devices is getting popular.

According to Gartner [4], the enterprises that have deployed edge solutions in the production are anticipated to extend from about 1% at the end of 2018 to about 40% by 2024 [5]. The worldwide edge spending guide from IDC states that the market for edge devices will reach $250.6 billion in 2024 [6]. A report from Tractica [7] predicts that the worldwide shipment of edge devices from 161.4 million units in 2018 will increase to 2.6 billion units annually by 2025 [8].

In the battle of edge AI, Nvidia, Google and Intel are three major players that are providing the support to harness the power of AI at the edge. by offering essential hardware platforms.

Initially, Raspberry Pi was widely used as an edge device in IoT paradigm for several years. It was a SBC launched by Raspberry Pi Foundation in 2012 for educational purposes. Its moderate speed processor and support for peripheral interfacing and networking made it well-suited for IoT-driven operations such as data sensing [9]. However, it was a limited solution because it does not come with a GPU. All its tasks are performed by the processor which is not efficient.

More recently, NVIDIA has come up with different GPU accelerated boards for driving AI innovation at the edge. The embedded computing boards of the Nvidia Jetson series (Nano, TK1, TX1, TX2, Xavier, and AGX) depict various capabilities and tradeoffs between costs and performance. Nvidia released the Jetson TK1 in late April 2014, whereas TX1 in 2015 and TX2 in March 2017 as AI-enabled edge devices. Later, Nvidia introduced Jetson Nano 4GB in March 2019 and Jetson NX at the end of August 2018 while Jetson AGX Orin in the first quarter of 2022 [10] for processing deep learning applications and AI data for edge computing.

Apart from the improvements in hardware platforms of edge devices, their limited computation and memory resources [11] are the challenges that affect DL-based models inference. The reason is, models trained with different DL frameworks such as TensorFlow [12], Caffe [13], PyTorch [14], Darknet [15] and PaddlePaddle [16] are not designed for low-powered devices. Therefore, the execution of these models on the edge devices hinders their performance. This has led to the development of inference compilers [17] that include Relay [18], TensorRT [19], TensorFlow Lite [20] and TVM [21] which optimize the inference of DL-based models for deployment at the edge devices. Among them, TensorRT from Nvidia and TFLite from Google provide state-of-the-art solutions. TFLite was initially developed for deploying the DL-based models on the iOS and Android devices (i.e. tablets, mobile phones), whereas TnesorRT was designed for accelerating inference on Nvidia GPUs. These compilers take the models developed in DL frameworks [12], [13], [15], [16] and optimize them for real-time inference on the edge devices.

Meanwhile, the spike in edge devices has motivated the researchers to develop and run DL-based models with real-time inference on the edge devices for new robotic applications in different fields. The use of robots has come into the spotlight with the Covid-19 pandemic. It is an infectious disease that spreads from person to person mainly through droplets of saliva or runny nose. The virus is transmitted when an infected person coughs or sneezes. Wearing the face mask has been recommended [22] to reduce the virus transmission and save lives since the initial outbreak [23]. Therefore, many countries have enacted laws mandating to wear face masks in both indoor and outdoor public places such as shops, schools, hospitals, airports and open area gatherings.

However, as humans, we are prone to forgetfulness and asking someone to wear a face mask is a touchy subject. Therefore, a safe alternative is to develop adaptable robotic solutions to monitor and remind the people to wear a face

mask. A promising avenue in this Covid-19 situation is the use of service robots. According to the new report of Polaris Market Research [24], it is anticipated that the size of global service robotics market is estimated to reach USD 54.4 billion by 2026 with a Compound annual growth rate of 17.3% from 2018 to 2026.

Indeed, these service robots can be used in hospitals to improve the safety of frontline HCW, hotels, super markets, airports, universities and offices to ensure that people are wearing face masks and reminding them that wearing face masks is compulsory. In this direction, Kyoto-based research institute ATR deployed a robot nicknamed ''Robovie'' at a sports store to make sure its customers wear face masks [25]. The university hospital Pitié Salpêtrière in Paris lent 4 Pepper robots developed by SoftBankRobotics with a new feature of face mask detection during the pandemic [26].

In this direction, we present GuardBot, which is a service robot powered by Nvidia's Jetson Xavier NX, an embedded edge computing board to deploy our DL-based face mask recognition framework with voice interaction for assisting the humans in preventing them from getting infected by Covid-19. The task is accomplished when GuardBot's camera scans the faces of the people and identifies the people without masks. A polite reminder/ warning is carried out through a generated voice message.

A. CONTRIBUTION

The following are key contributions of our work.

1) For face detection module, we pursue two main goals:
   i) We first draw comparisons of traditional and deep learning based face detectors. The aim of this comparison is to select the one detector that is the most accurate in terms of multi-view face detection.
   ii) After obtaining the most accurate face detector, our next objective is its optimization that improves its inference speed on the edge device that is Jetson Xavier NX in our work.
2) For face mask recognition module, we proceed with the following goals:
   i) We apply transfer learning on seven pre-trained models and then we fine-tune them with the same parameters during training. In addition, we contribute to optimize all the models for edge specific inference.
   ii) We also proposed our CNN model architecture with optimized parameters which has been compared with transfer-learning based custom CNN models. In our contribution, we focused on minimizing network complexities and simultaneously maintaining its accuracy. To accomplish our goal, we first fine-tune our network (discussed in Section Framework III) during the training phase and then optimize the fine-tuned model for obtaining real-time inference speed.
   Our proposed CNN-based classifier for face mask recognition is trained from scratch, which is another contribution. The main advantages of our proposed architecture are:
   - A considerably good accuracy for face mask classification task.
   - A highly reduced complexity due to its slim architecture compared to other state-of-the-art models.
   - The minimum risk of negative weight transfer due to model training from scratch.
   - The usability of the model in computationally resource-limited platform.
   - High throughput and low latency.
3) We integrate visual recognition modules with the voice alert system for generating a warning voice message if a person without a mask is identified.
4) Finally, we deploy our proposed framework on GuardBot that is equipped with Jetson Xavier NX and Zed camera. It performs face mask recognition task in patrolling or non-patrolling modes. We access its performance in both outdoor and multi-floor indoor environments.
5) Moreover, we present performance comparison based on the results of all these models after training-testing phase and optimization phase.

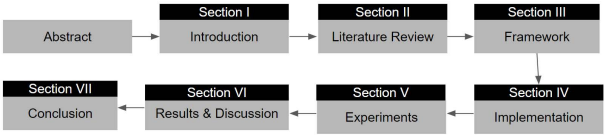

FIGURE 1. Paper structure.

The overall structure of our paper is shown in Figure 1. The rest of our article is organized as follows: Section II presents literature review related to edge computing and optimization of DL-models. It also discusses the previous studies related to face mask detection with a brief review of speech recognition work. Section III gives a detailed description of our proposed two-stage framework that consists of three main modules. It also explains the architecture of each DL-based model used in our framework. Section IV covers implementation details of all the models and techniques that we have used in our framework. Section V describes the hardware and software requirements for conducting the experiments in real-world scenarios and presents two case studies. Section VI presents the results and discusses the performance at the edge deployment using different evaluation metrics. Finally, we conclude our work in Section VII with an outline of future research direction.

II. LITERATURE REVIEW

We have divided the literature review section into three subsections. In the first subsection II-A, we first discuss the work related to edge computing in comparison with the the cloud computing. In the second subsection II-B, we review the optimization approaches for edge devices while in the

third subsection II-C, we discuss the background work related to the face mask II-C1.

A. EDGE VS CLOUD COMPUTING

Deep learning [1] has emerged as a particular form of hierarchical learning that uses multi-layer neural networks to learn efficiently and progressively the useful data representations through the extraction of higher-level features from the raw input. It has demonstrated promising results and brought revolution in several applications related to the robotics [27], object detection and recognition [28]–[35], image classification [36]–[38], character recognition [39], [40], document classification [41], NLP [42], speech recognition [39], [41], [43], human pose estimation [34], activity recognition [32], [41], brain-computer interface [44], medical applications [45] and autonomous driving [46].

Its prediction algorithms, also known as models, have been consistently outperforming traditional methods for the last few years [47], [48]. However, high accuracy of these models comes at the expense of high computational resources [49] and memory requirements.

A common approach to overcome the computational resources is to leverage cloud computing [50] in which data of IoT sensors is transmitted from the source to a centralized location. It introduces challenges related to the latency [51], scalability and privacy [52] that are critical for many applications. For example, camera frames of AV require to be swiftly processed for detection and obstacle avoidance or voice commands given to a robot need to be instantly parsed for returning the quick response to the user's query. However, cloud-based data processing cannot satisfy the end-to-end low-latency requirements of these applications due to additional queuing and propagation delays. Mahadev *et al.* [53] showed that camera frame offloading to AWS server for the execution of end-to-end vision task takes more than 200 milliseconds. The cloud-based bandwidth-intensive data sources such as video streams have the limitation of scalability for a large number of streams that is particularly a concern for most of the commercial systems such as cloud-based video surveillance. It also exacerbates security and privacy issues which increase the risk of data leaks. For example, a study [54] reported that 40% of business organizations experienced cloud-based data breach. A study discussed in [55], stated that resource sharing in cloud robotics leads to 75% of security challenges.

Edge computing [55] is a feasible solution to overcome these issues related to the latency, scalability, and privacy challenges by bringing the computation and data storage closer to the source of the data itself at the end devices. In recent years, it has received considerable attention in the academic and industrial circles. It has become a key enabler for many technologies where rapid response to the sensor is necessary. For example, in the 5G many applications rely on edge computing to perform real-time interaction and local data processing [56], In IoT [57], high bandwidth and reduced end-to-end delays were achieved by proposing a multi-access virtualized edge computing framework [58]. In AV, it provides support to design an edge computing ecosystem with enough computing power and security to guarantee the safety of AV [34]. Similarly, in the field of healthcare service robots, edge-based solutions are preferred in a situation where a surgeon using a medical robot can not tolerate latency and network delays [59], It is envisioned to handle different applications in human-robot interaction [60], [61]. In [60], edge computing based robot interface was proposed for automatic mental health care of elderly people using a programmable embedded device with audio processing unit and environment noise filtering to understand voice commands. In [61], a framework for guide dog robot was presented based on an edge computing on Intel UpSquared board and a prototype was developed for obstacle avoidance and traffic sign recognition along with human following and voice recognition.

B. OPTIMIZATION FOR EDGE DEPLOYMENT

In the era of edge computing, equipping the edge devices with AI powered deep learning models has attracted the attention of many researchers and companies for providing the real-time solutions for deployment. However, despite the edge computing advantages of low latency, scalability and privacy, the deployment of the DL-based models on the edge devices is still a major challenge in terms of computation, memory, and power consumption [62]. Typically, once the model is trained with the popular DL-based frameworks such as TensorFlow [12], PyTorch [14], Darknet [63] or Caffe [13]. it is typically used for inference in the same framework. However, the model's slow speed and high latency might cause performance issues when the model is run using [12]–[14], [63] frameworks on the edge devices because it contains several additional layers that are not required for inference. Moreover, the trained model with these frameworks [12]–[14], [63] by default uses FP 32-bit precision, which makes it computationally expensive during the inference on the edge devices and as a result its performance decreases due to limited computational resources. To amortize the DL-based resource-intensive [64] networks for fast prediction (i.e., inference), the researchers have explored several techniques such as quantization [65]–[68], mixed precision inference [69] and weight pruning [64], [70]–[72]. In this direction, different optimization frameworks have also been proposed for mobile, embedded and edge devices [72]. Among them TensorRT [73] from NVIDIA and TFLite [20] from Google are the most popular and state-of-the-art framework because they encompass most optimization techniques which make it possible to execute DL-based models on a single accelerator.

TensorRT [73] is built on the top of CUDA that performs various optimizations to generate high throughput and low latency for DL-based models on the Nvidia GPUs and Jetson boards. It optimizes the models trained in other frameworks. Besides the mixed-precision computation, it calibrates the weights of trained models along with its activation function. It performs quantization for precision reduction to FP32,

FP16 and INT8 which produces high throughput. Furthermore, It also performs layer and tensor fusion for memory optimization. In a study [74], face recognition model using TensorRT and TensorFlow Lite was deployed on different GPUs (GTX 1080, RTX 2080Ti, RTX 2070, and RTX 8000) and edge devices. The experimental results showed that TensorRT optimization had the fastest inference speed compared to TensorFlow Lite on all GPUs and edge devices at the expense of energy consumption. In another study [17], performance of TensorFlow-TensorRT (TF-TRT) and TensorFlow Lite (TF-Lite) was analyzed on the edge platform by comparing their throughput, latency and power consumption. It showed that TF-Lite performance was better with light weight TF-models than the deep network while, TF-TRT performance was consistently higher with different DL-based models.

C. BACKGROUND

In this subsection, we first discuss the approaches that have been used for face mask recognition and after that we compare our work with the related work for face mask detection.

1) FACE MASK RECOGNITION

COVID-19 is the infectious disease caused by SARS-CoV-2 that emerged in December 2019 [75]. According to the CDCP [76], direct or indirect contact and droplet spray in short-range are the reasons that cause its spread [77]. The SOP's by WHO suggest the best way to prevent and slow down COVID-19 transmission is to wear the face mask [78]. However, it is a challenging task to monitor the people manually in crowded indoor or outdoor places. Researchers have attempted different approaches to automate this task. Prusty *et al.* [79] applied filtering techniques during the preprocessing step to perform data augmentation using grayscale [80] images with gaussian blur [81] and detected the face mask using YOLOv3 [82]. Wang *et al.* [83] proposed ''WearMask'' application to provide web based solution for face mask detection using YOLOv3 [82] while performing the optimization using NCNN [84] framework. It used MAFA [85], RMFD [86], MMD [87] and SMFD [88] datasets for faces with masks and WIDER FACE [89] dataset for faces without mask.

A study [90] presented the hybrid model consisting of two components. Its first component was designed for face feature extraction using ResNet-50 [91] while the second component was face mask classification using decision trees, ensemble algorithm and SVM. It used the SMFD [88], the LFW [92] and RMFD [86] datasets while highest accuracy of 99% was achieved on the LFW with SVM. This model produced high accuracy, however its speed was slow. The practical application for deployment in a productive environment requires a model to work in real-time on the edge or portable devices. Putro *et al.* [93] used face RoI and developed a FFDMASK to detect the faces and built a CNN architecture with attention module for mask classification on Jetson nano. It used a wider face [89], SMFD [88] and LFW [92] datasests.

In another study, Nagrath *et al.* [94] proposed SSDMNV2 approach. It detected the faces using SSD [95] and performed the face masks classification using MobileNet-v2 [96] architecture. Militante *et al.* [97] performed face mask classification using VGG-16 on Raspberry pi. It set an alarm if the person is not wearing a face mask and created a knowledge-based dataset.

Table 2 reports our model's comparative characteristics with the previous related work.

III. FRAMEWORK

In this section, we discuss the overall architecture of our framework for face detection, mask recognition and optimization modules. The pipeline of our framework's deployment architecture is illustrated in Figure 2.

Our two-stage framework architecture consists of three main modules. The first module performs face detection with the MTCNN at the first stage and the second module involves the face mask recognition using our proposed CNN model. It also contains transfer learning based seven custom CNN models at second stage. While the third module performs optimization to execute real-time inference for all models on Jetson Xavier NX for deployment on the GuardBot. It generates the voice message when detected person is not wearing a face mask.

In the following sections, we explain three module and discuss the models that have been used in these modules.

A. STAGE 1: FACE DETECTION MODULE

The first stage of our architecture contains a face detection module that has been deployed using MTCNN [98] face detector. Although, for face detection, we have implemented four pre-trained face detection models on Jetson Xavier NX which are Haar cascade [99], HOG [100], Caffe DNN [13] and MTCNN [98] face detectors. However, the objective of implementing four face detectors is to select the one model which performs better among all in terms of accuracy and then optimize the most accurate detector for real-time inference on the edge device. As a result, among all these, we have selected MTCNN for optimization and deployment due to its better accuracy compared to other models [as compared in section VI-A].

1) HAAR CASCADE FACE DETECTOR

It [99] is a ML-based approach to train the classifier using a lot of positive (images of faces) and negative (images without faces) examples. It consists of four steps.

- Haar Feature Selection
- Creating an Integral Image
- Applying Adaboost [101]
- Cascading Classifiers

For face detection, it applies haar rectangular features which are similar to the kernel convolution and uses an image representation called integral image to evaluate the rectangular features in constant time. As a result, each feature returns

TABLE 2. Characteristics of our work compared with the work related to Face Mask Recognition.

Ref #	Strategy	Dataset	Face Detection Model	Mask Recognition Model	Voice Message	Framework	Optimization	Deployment
[77]	One stage	Kaggle	YOLO-v3		No	*	No	*
[81]	One Stage	WIDER FACE, MAFA, RMFD, MMD	YOLO-v3		No	TensorFlow	Yes (NCNN)	Web-based
[95]	Two Stage	self-made	*	VGG-16	Yes	*	No	Raspberry pi
[88]	Two Stage	SMFD, LFW, RMFD	ResNet-50	Decision Tree, Ensemble Algo, SVM	No	*	No	*
[91]	Two Stage	WIDER, FACE, SMFD, LFW	FFDMask	Slim CNN	No	Keras, PyTorch	No	Edge-based (Jetson Nano)
[92]	Two Stage	Self-made, Different online resources	Single Shot Multi Box	MobileNet-v2	No	TensorFlow, Keras	No	*
Our Work	Two Stage	Kaggle	MTCNN	Our CNN, MobileNet-v2, Inception-v3, VGG-16, ResNet-50, DenseNet-121, Xception, NASNet-Mobile	Yes, (PyAudio)	TensorFlow, Keras	Yes, (TensorRT)	Edge-based (Jetson Xavier NX)

* It is not explained in the paper.

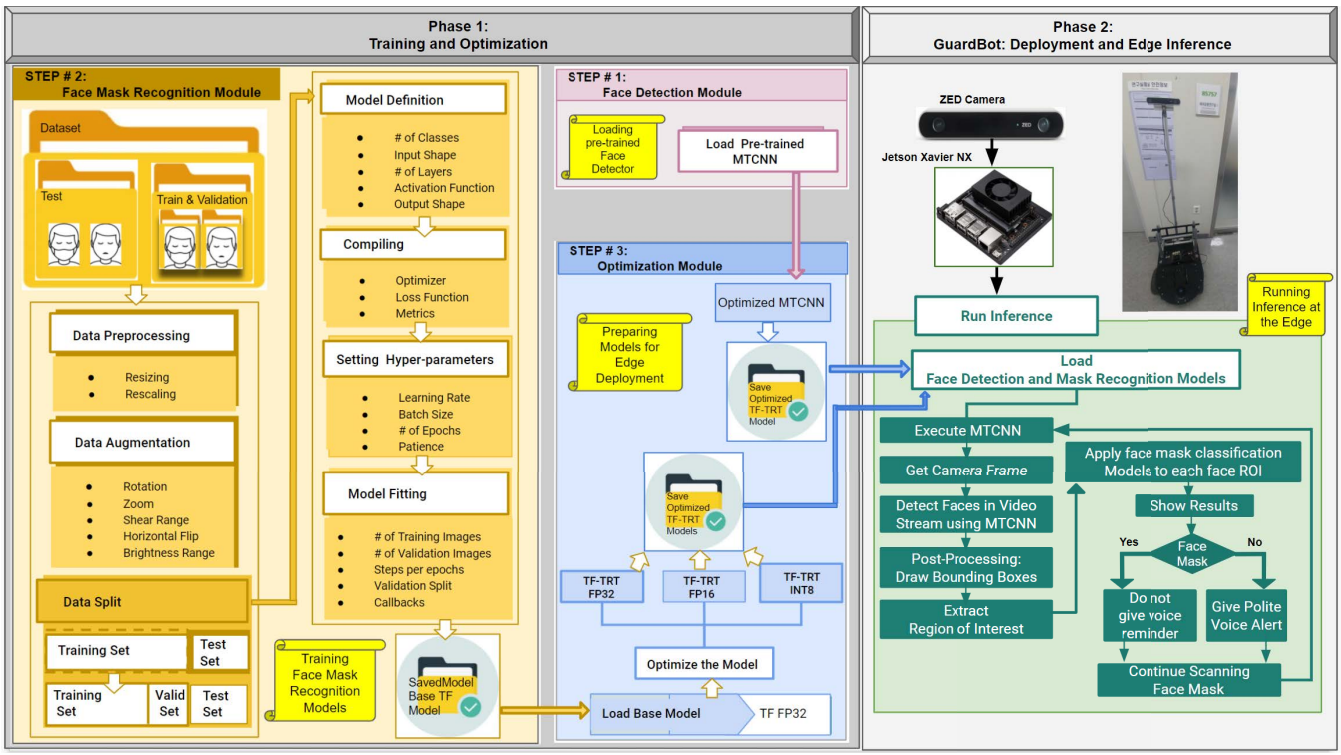

FIGURE 2. Flow diagram of our framework.

a single value that is calculated by subtracting the pixels' sum of white rectangles from the pixels' sum of black rectangles as:

$$\text{value} = \sum(\text{pixels in black area}) - \sum(\text{pixels in white area})$$

Its object detection procedure employs a variant of the learning algorithm AdaBoost [101] for the selection of best features and training of the classifier (30). Its attentional cascade technique combines complex classifiers into cascade

structure for focusing on the promising region of the image to improve the detection.

2) HOG FACE DETECTOR

The HOG [100] is a feature descriptor. The idea is to extract the face features into a vector using HOG. The features are described by distribution (histograms) of edge directions and intensity gradients. The gradients are large around the corners and edges which help in detection. When we pass an image to HOG, it divides the image into cells and computes the histograms of gradient directions for the pixels in each cell. These histograms represent the descriptor. The descriptor takes an input image of size 64 $\times$ 128 $\times$ 3 and the length of its output feature vector is 3780.

3) CAFFE DNN MODEL

We have used the Caffe DNN model for face detection. Caffe models were developed as a fast and efficient alternative to other frameworks for performing object detection and recognition. Caffe DNN model. Caffe DNN model is based on the SSD (Single Shot-Multibox Detector) [95] and uses ResNet-10 architecture as its backbone.

4) MTCNN FACE DETECTION

The MTCNN [98] face detector performs both detection and alignment tasks using multi-task learning. It adopts a cascaded structure for landmark prediction and face detection. Its architecture consists of a three-stage network. At first stage, it uses P-Net which is a fully Convolutional network. It is used to obtain candidate windows and their bbox regression vectors. At the second stage it uses R-Net which is a convolutional neural network. It is used to refine the bboxes by performing calibrating with bbox regression and employing NMS (Non-Maximum Suppression). At the third stage, it uses O-Net which aims at describing the face and its landmarks with more precision. As a result, it returns three outputs: Facial landmark localization, BBox regression, and face or not face classification.

B. STAGE 2: FACE MASK RECOGNITION MODULE

In the second stage, for face mask recognition, we have proposed our own CNN model for face mask classification. In addition, we have applied the transfer learning based approach to seven pre-trained CNNs for building custom models that are: MobileNet-v2 [96], Inception-v3 [102], VGG-16 [103], ResNet-50 [91], DenseNet-121 [104], Xception [105], NASNet-Mobile [106]. All these models take RoI from the face detection module as input and their outcome is face mask recognition based on the classification. In the following section, we discuss the architecture of our proposed CNN model along with seven TRL-based custom models.

1) OUR PROPOSED CNN MODEL

In this work, we have built our own classification CNN for the face mask recognition task to classify input images without relying on pre-trained models. The purpose is to design light weight and less complex model for reducing the image dimensionality and identifying the patterns related to each class.

We have started to develop the custom model in TensorFlow with the functional API of Keras because it is more flexible than sequential API. It is a way to build graphs of layers and allows the creation of layers in DAG. In our model definition, the initial size of the input image to our network is 224 $\times$ 224 with three channels. Our model takes a color image as input which is scaled to 224$\times$224$\times$3 for training and testing. Figure 3 represents the overall definition of our model.

The architecture of our model consists of four convolutional blocks in which each block contains one convolutional layer followed by a ReLU layer and a max pooling layer. The first convolutional block uses 16 filters of size 3 by 3. Each filter slides across the whole image and produces a separate 2D activation or feature map as output using ReLU activation function. Max pooling of size 2 by 2 is then used to reduce the spatial dimensions of the output volume. We stack three more convolutional blocks in which the second, third and fourth convolution blocks are stacked with 32,64 and 128 filters. Each convolutional block has a higher number of filters than the previous one.

To make the prediction, we need to convert the output feature maps of convolutional blocks into a single or one-dimensional array. For this, we have used a Flatten layer in our model which gets a multidimensional output and makes it linear to pass it as input to the dense layer.

We add two dense layers with 256 and 50 neurons. These layers are also referred to as the fully connected layers. Each FC layer has been followed by two dropout layers (dropout ratio value 0.2) to constrain the network from over-learning. Adding more than one FC layers, prior to the layer that makes predictions, allow greater coverage for the entire spatial dimension of the image and better interpretation between the features extracted by the convolutional blocks and the predictions.

Finally, we construct a prediction output layer which is a dense or FC layer with softmax activation function. The reason for using Softmax instead of Sigmoid is to generate two output neurons from our model instead of one. Hence, our final dense layer has the same number of output nodes as the number of classes. It outputs target class probabilities where each value ranges between 0 & 1 and all values sum to 1. Finally, we group input and output layers into the model object.

2) TRANSFER LEARNING-BASED CUSTOM MODELS

I) MobileNet-v2

MobileNet-v2 [96] is lightweight in its architecture which seeks to perform well on mobile and embedded edge devices. It introduces inverted residual structure which consists of thin bottleneck layers for residual block's input and output. It performs depth-wise separable convolutions to extract the features from each

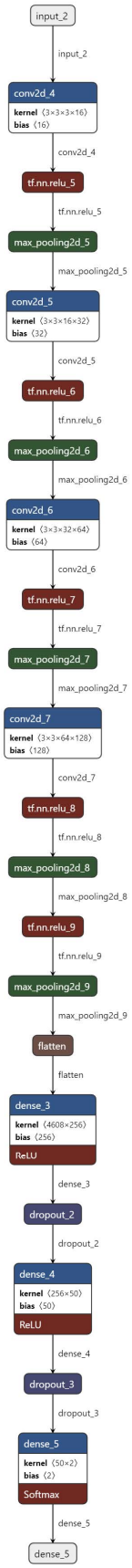

FIGURE 3. **Our proposed CNN model for face mask recognition based on classification.**

color channel. It maintains the representational power by removing the non-linearities in n-array layers. Its architecture contains two main blocks. One residual block is with stride of 1 while second is with stride of 2 for downsizing. Both blocks consists of three layers. The first layer is 1×1 convolution with ReLU6 [107], second is depth-wise convolution and third is 1×1 convolution without any non-linearity. The main intuition is to use the bottlenecks for encoding inputs and outputs while the inner layer for encapsulating the transform from lower to higher-level i.e. from pixels to image categories [108].

II) Inception-v3

It [102] focuses on the less computational power by introducing the idea of factorized convolutions: It aims at reducing the number of parameters or connections without affecting the network efficiently. For this, it performs factorization into convolutions by replacing large convolutions with smaller. For example, two 3×3 convolutions ($3 \times 3 + 3 \times 3 = 9+9=18$) replace one 5×5 convolution ($5 \times 5=25$) which results in parameter reduction from 25 to 18. Similarly, it also performed factorization into asymmetric Convolutions. It also adds an auxiliary classifier that acts as a regularizer.

III) VGG-16

The VGG-16 [103] architecture is an improvement over AlexNet. Its idea was to replicate large size convolution filters ($11 \times 11, 5 \times 5, 3 \times 3$ used in Alexnet) with multiple 3×3 filters of a fixed size. Its 16 layer architecture has a total of 138 million parameters consisting of 13 convolution and 3 fully connected layers. Instead of having a large number of hyper-parameters, VGG-16 focuses on convolution layers with stride of 1 and max pooling layers with the stride of 2. Its whole architecture follows the same arrangement for convolution and max pooling layers.

IV) ResNet-50

ResNet-50 [91] is much deeper than VGG-16 [103], however, its size is substantially smaller because it uses global average pooling instead of fully-connected layers. It uses skip connections and batch normalization to train deep layers without affecting the model's performance. The deep CNNs are difficult to train due to the vanishing gradients problem. However ResNet-50 provides its solution by introducing skip connections which are also known as gated units. These gated recurrent connections allow to train the model of 152 layers with lower complexity than VGG-16 [103]. It runs residual functions using $H(x) = F(x) + x$ for a few stacked layers instead of learning the direct mapping. It contains 23 million parameters which makes it faster to train compared to the VGG-16 [103]. The ResNet-50 architecture is based on the Resnet-34 with one major difference. That is a modified building block into a bottleneck design which uses a stack of 3 layers instead of 2. The skip connections do not contain extra parameters

and cause less computational complexity and allow to transfer relevant detail from previous to the next layer.

V) DenseNet-121
The DenseNet-121 [104] architecture is the modification of standard CNN architecture. The DenseNet-121 architecture contains one 7 × 7 Convolution, fifty eight 3 × 3 Convolution, sixty one 1 × 1 Convolution, four average pooling layers and one FCN layer. Connectivity, dense blocks, growth rate and bottleneck layers are main components in its architecture. In CNN the feature map of the previous convolutional layer is passed to the next layer. In this way, CNN provides L direct connections for L layers. In contrast, the main idea of DenseNet-121 is the concatenation of feature maps from previous layers such that for L layers there are L(L+1)/d direct connections. Instead of extremely deep or wide architecture, The DenseNets exploit the potential representational power through feature reuse and alleviate the vanishing-gradient problem.

VI) Xception
Xception [105] stands for ''extreme inception''. It is inspired by the Inception model to improve multi-scale feature extraction. It combines the advances from ResNets [91] and Inception [102]. The design of the Xception model is based on the idea of depth-wise separable convolution. The Xception model is separated into 3 main parts which are main flow, middle flow with 8 repetitions of the same block and exit flow. It contains 36 convolutional layers which are structured into 14 modules with linear residual connection, except for first and last modules. All Convolution and SCL (Separable Convolution layers) are followed by batch normalization

VII) NASNetMobile
NASNetMobile [106] stands for Neural Architecture search (NAS) Network. It is a scalable CNN architecture which consists of cells that are its basic building blocks and have been optimized using reinforcement learning. A cell performs some operations such as separable convolution and pooling. These operations are repeated according to the network capacity [109]. The mobile architecture of NASNet contains 12 cells and 5.3 million trainable parameters.

C. OPTIMIZATION MODULE

Optimization is an additional but important step that is performed before deploying the models for inference.

In this module, we optimize the stage 1 (section III-A) and stage 2 (section III-B) models to speed up the inference on the Jetson Xavier NX. We accelerate the inference using the Nvidia TensorRT integration with TensorFlow to obtain maximum performance at run-time with low latency and high throughput of our DL-based models. TensorRT is a library that has been developed by Nvidia to accelerate the inference process on Nvidia GPUs and Jetson boards. It is built on CUDA that supports parallel computing platforms to use the computing capabilities of GPUs. TensorRT allows optimizing the model trained in different deep learning platforms (i.e. TensorFlow, Caffe, PyTorch) and performs lower precision on model weights while maintaining the accuracy for final deployment of models to a variety of environments.

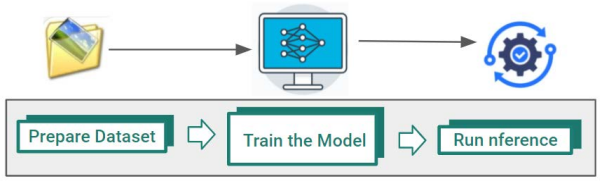

FIGURE 4. Typical workflow to perform inference using TensorFlow.

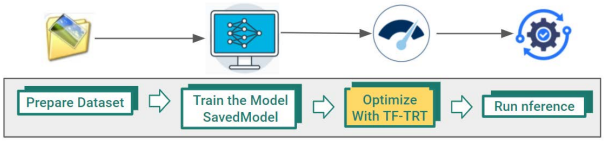

FIGURE 5. Optimzed Workflow to perform inference using TensorFlow-TensorRT via ''SavedModel''.

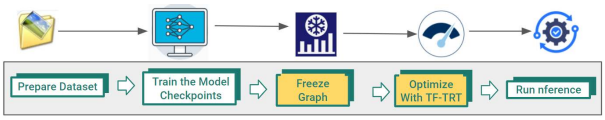

FIGURE 6. Optimzed workflow to perform inference using TensorFlow-TensorRT via ''frozen graphs''.

The typical workflow when deploying the DL-based TensorFlow trained model for inference is shown in Figure 4.

However, to optimize the DL-based models in TensorFlow with TensorRT integration (TF-TRT) the workflow changes in one of the two forms as shown in Figure 5 and 6.

The inference is performed using the saved model if the trained models are stored in the ''SavedModel'' format as illustrated in Figure 5 or it runs using frozen graphs if models are saved with regular checkpoints as described in Figure 6.

In our study, all custom trained TensorFlow models have been saved in ''SaveModel'' format, therefore our optimization workflow follows the Figure 5. TensorFlow employs graphs for ''SavedModel'' format when it exports them from Keras. TensorRT integration applies the optimizations to compatible sub-graphs that consist of TensorFlow with TensorRT (TF-TR) supported operations [110]. For this, it scans all the sub-graphs that can be optimized based on the operations supported and replaces them with TensorRT optimized nodes while it leaves the unsupported operations untouched which are handled natively by TensorFlow.

D. VOICE MESSAGE ALERT SYSTEM

GuardBot' camera detects the faces of people approaching it at stage 1 (section III-A), and if it detects their face is uncovered at stage 2 (section III-B). A call to the voice message alert system is sent that pronounces the phrase: ''You have to always wear a mask.''

IV. IMPLEMENTATION

In this section, we describe the implementation details of our framework.

A. FACE DETECTION MODULE

This section discusses the implementation of four pre-trained face detection models on Jetson Xavier NX that include Haar cascade [99], HOG [100], Caffe DNN [13] and MtCNN [98] face detectors.

In our framework, we have selected MTCNN because of its accuracy (section VI-A) and used its optimized model (Section IV-A) for real-time face detection at the edge device.

We apply Haar cascade to the images and videos using the OpenCV library. To detect faces using Haar cascade, we draw rectangles around the detected faces on the color images by performing following steps:

- Step 1: We load the frontal-face xml detector
- Step 2: We resize the image to 224 × 224 and input it by converting into grayscale because Haar cascades work only with gray images.
- Step 3: We run the classifier to detect the faces.
- Step 4: We draw the rectangles around detected faces and extract the coordinates.
- Step 5: We create an object called faces and store the detected faces in it.
- Step 6: We use detetMultiScate() function to obtain the tuple of four elements from the rectangle: x and y coordinate of top-left corner, and w and h which are width and height values.
- Step 7: We set the scale factor to default that is 1.05. For reducing the image size at each scale.
- Step 8: We specify the minimum number of neighbors to 5. For determining the number of neighbors each candidate rectangle should have to retain it.

HOG implementation consists of five main stages.

- Pre-processing
- Gradient computation
- Calculating magnitude and orientation
- Calculating Histogram of Gradients (8 × 8 cells)
- Normalize gradients (16 × 16 cells)
- Calculate the final feature vector of complete image

The first step is image preprocessing and resizing it to the width to height ratio of 1:2. Although, the descriptor can take the images of different sizes with an aspect ratio of 1:2. However, it should preferably be 64 × 128. Because we will divide the image into 8*8 and 16*16 patches for feature extraction. Having the specified size (64 × 128) will reduce the computational cost (43). In this step, the performance gains of gamma corrections are minor. It can be skipped.

In the second step, we calculate the gradient of each pixel which represents the change in x and y directions. To calculate it for the selected pixel in x direction, we subtract left pixel value from the value on the right, while for y-direction we subtract below pixel value from the above pixel value. We achieved it by filtering the image using kernels. Same results can be obtained using the Sobel filter [111] with kernel size 1.

In the third step, we determine the magnitude and direction of gradient for all pixel values using the following formula.

$$g = \sqrt{gx^2 + gy^2}$$
$$\Theta = arctan(\frac{g_y}{g_x})$$

The higher magnitude value represents sharp change in intensity around the edges.

In the 4th step, we divided the image into 8 × 8 cells to calculate a 9-bin histogram of gradients for all 8×8 cells. The gradient for individual pixels contains 2 values (magnitude and orientation) which accumulate $8 \times 8 \times 2 = 128$ numbers. These 128 numbers are stored in 9-bin histograms corresponding to angles 0, 20, 40, 60 . . . 160. Each cell is weighted using gradient direction and voted (the value that goes into the bin) into 9-bins based on the magnitude. Finally, all voted values are added up to create the 9-bin histogram.

In fifth step, the image gradients calculated in the last step are sensitive to light. In this step, we normalize the gradients by taking 16 × 16 blocks to reduce lighting variation. Initially, we combine 4 adjacent cells in which each contains a feature vector of size 9. Consequently, we create a concatenated feature factor of size 36 (4 cells * size of feature vector in each cell). To normalize this 36×1 matrix, we divide each vector value by the square root of the sum of squares of the values.

In the last step we obtain the final feature vector for a grid of 8×16. It ends up with 7 horizontal and 15 vertical (16×16) blocks where each block is represented by a 36 × 1 feature vector. We concatenate all these vectors and obtain a single (7 × 15 × 36 =) 3780 dimensional vector of the complete image.

Caffe DNN model introduced in the deep neural network module of post OpenCV 3.3 has been implemented. We have used the 'readNetFromCaffe' method to load the pre-trained model with the OpenCV library. For this, we have downloaded pre-trained model weights and prototxt files which were passed to the method as arguments. After some preprocessing, we pass the images and video frames to the model to get the bboxes of detected faces.

MTCNN is very complex model for face detection. However, its third-party open source implementations are available under a permissive MIT open source license which save time and allow to train MTCNN with new dataset or use a pre-trained model for face detection. Perhaps the best third-party Python-based MTCNN are availabe by Iván de Paz Centeno [112] with Keras and OpenCV support, jbrownlee [113] with TensorFlow and OpenCV support and linxiaohui [114] with OpenCV support. In our work, we have used TensorFlow and OpenCV implementation of jbrownlee [113] that is a forked from [112]. It uses FaceNet's MTCNN [115] as source which is based on the work of [98] from 2016.

Optimization: MTCNN is the state-of-the-art approach for face detection with high accuracy (section VI-A1 among HOG, Haar cascade and Caffe DNN model. That is the main reason of selecting MTCNN for deployment on GuardBot. However, embedded device's capabilities to efficiently process the frames with real-time FPS rate is a challenge in the edge deployment. The incoming frames processing with MTCNN is slow on Jetson Xaveir NX (section VI-A2) due to limited computational power of the embedded board. To overcome this issue, we have used its TensorRT implementation from [116]. For this, we have customized the code of MTCNN TensorRT for the integration with our face mask recognition module.

B. FACE RECOGNITION MODULE

In this section, we first discuss the dataset and its split that we have used for training and evaluating the models. After that, we explain the implementation of our proposed CNN classification model to recognize the face mask. It is then followed by the implementation details of the TRL-based approach for fine-tuning the seven models ([102], [91], [103]–[105], [106]) to classify face mask images for recognition. At the end, we elucidate optimization of these models.

1) DATA SPLIT

All images belonging to two classes have been divided into three train, validation and test data splits using ImageDataGenerator class that makes 700 images for testing the models' performance, 2260 images for their validation and 9040 images for training the models. Initially, we used a learning rate of 0.001 for models before fine-tuning. However later, it was set at 0.0001 for fine-tuning both our model and transfer-learning based custom models. The batch size of 32 with 50 epochs has been used during training the models. We avoid the models' over-fitting of the training dataset using early stopping which restores the best weights with a patience of 10 by monitoring the validation loss. The cross-entropy has been used as a loss function to compare the predicted probabilities with actual class output values between 0 and 1.

2) PROPOSED CNN MODEL

The end-to-end pipeline of our proposed CNN model is described below.

- Pre-processing and Augmentation:
  Data preparation is the first step after data collection. In this step we have performed image pre-processing, data normalization and augmentation steps. In the pre-processing step, we standardize the shape of the input images with the same width and height (224×224). Reducing the size of images helps to decrease the training time and increase the inference speed without significantly affecting the performance. We also normalize the image data by scaling the pixel values to the range 0-1 before passing the data to CNN layers. Our data augmentation step involves flipping images horizontally, applying shearing transformations, randomly zooming inside the image, turning by increasing or decreasing the brightness level. This step artificially creates different versions from existing data to allow our model to generalize across images trained on different flips, shear, zoom and lighting levels. We apply pre-processing steps both on training and testing images while augmentation steps only on the testing image
- Compile the Model:
  After defining the model architecture (section III-B1), we compile it by specifying the optimizer, loss and metrics. We specify loss function as category cross entory to evaluate how well the network models the dataset during network tracing. It outputs a lower number if predictions are good, otherwise it returns a higher number.
  We have used accuracy, recall, precision and auc metrics to determine the performance of the model. Unlike loss function, these metrics are not used for training the network. We instantiate the Adam optimizer which is an adaptive learning rate optimization algorithm with the suggested [117] learning rate (0.001). We change the learning rate (0.001) during the fine-tuning process,to reduce overall loss and increase the accuracy.
- Training:
  Next, we start training our model by specifying 50 epochs to train our network along with the batch size 32. We set up the data generators to draw the random batches of images from training and validation directories and generated the batches of data with augmentation that have been passed to fit the model during the training process. We have used callback techniques that allow automatic interaction with the network during training. EarlyStopping callback has been used to monitor the validation loss to find that it is decreasing. We specify a delay to ensure that it does not trigger at the first minimum value of the validation loss. Instead, it stops training after waiting for the 5 epochs with no improvement. The "ModelCheckpoint" callback has been used to save the best model automatically during the training process by monitoring the minimal validation loss on the dataset.
- Optimization:
  As a last step, we improve the model's throughput and reduce its latency by converting the trained model from TensorFlow to TensorRT. We have discussed its details in the Optimization Section IV-B4.
- Inference:
  Finally, we perform the inference on the Jetson Xavier NX and its results are discussed in the Section VI-B.

3) TRANSFER LEARNING-BASED CUSTOM MODELS

Our transfer learning based approach uses the layers, pre-trained on a source task and fine-tunes them to perform the target task that is face mask recognition. We downloaded seven pre-trained models which include MobileNet-v2 [96], Inception-v3 [102], VGG-16 [103],

ResNet-50 [91], DenseNet-121 [104], Xception [105], and NASNet-Mobile [106] models and cut off their top portion containing FC layer which leaves us with the convolution and pooling layers. We use these layers for feature extraction. Once the pre-trained layers have been imported and the ''top head'' of each model has been excluded, we employ the transfer learning approach on all the models for training with and without fine tuning techniques.

Our transfer learning approach is a multi-step process. For this,

- We first freeze pre-trained convolution layers in the network to ensure that previous generic features learned by the CNN are not destroyed.
- Then we remove the top head or last fully connected output layer(s) at the end of each network.
- After that, we create a new top head with five layers on the pre-trained layers of the model, which entails
  - One flatten layer: We add a new flatten layer that converts the output of earlier pre-trained layers from 2D filter maps into a vector.
  - Two dense layers: We include two dense layers consisting of 1024 and 512 neurons, followed by Relu activation. We pass the vector obtained from the flatten layer to the dense fully connected layers for classification.
  - One dropout layer: We append 0.2 dropout regularization to reduce the network over-fitting.
  - One output layer: We define an output layer with softmax activation to predict the probability for two classes. The total number of outputs of this layer is equal to the number of classes in the model.
- Finally, we group the input of the convolutional base and output of the new top head into a model object. It bootstraps the top model onto the pre-trained layers.
- Compiling: We compile the model by initializing the Adam optimizer, cross-entropy and metrics.
- Training: We train the compiled network with back-propagation.
  - Training without fine-tuning:
    To use transfer learning approach without fine tuning: We start training the network, but we only train the new top head of the model. For this, our new compiled model contains the pre-trained weights and layers of the model along with the new top head for face mask classification. In this case, we freeze all pre-trained layers. This model extracts the features using the frozen pre-trained layers and trains the output layer with suggested lr (learning rate = 0.001) to perform the predictions.
  - Training with fine-tuning:
    Alternatively, to entail the transfer learning approach with fine tuning in which some specific or all convolution layers of the pre-trained network are un-freezed and a second pass of training with the new top head of the model is performed. For this, we have re-compiled the models by un-freezing the last two pre-trained layers and used back-propagation to update the weights in the last two pre-trained layers with the new top head of the model. Then, we have trained this model with lower lr (learning rate = 0.0001) to warm up the FC layer in learning the patterns from previously learned convolution layers. The rest of our model with its top head and weights has been initialized just as before.

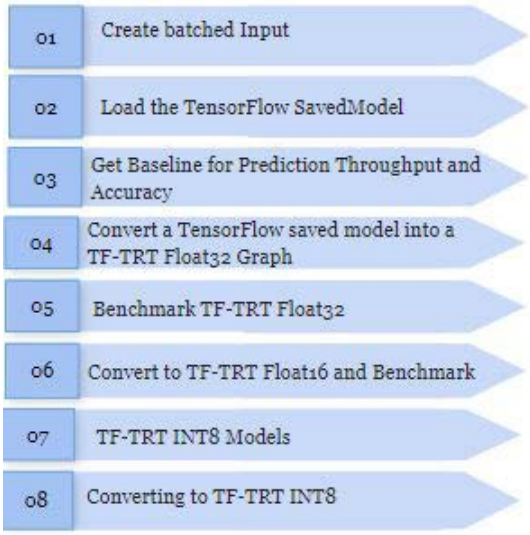

FIGURE 7. Optimization steps.

4) OPTIMIZATION

In this step, we perform optimization through quantization using TensorRT for all face mask recognition models to speed up their inference. The quantization is a process that reduces the precision of weights for all the models with the goal of improving run-time performance for the efficient inference. TensorRT supports two modes. The first is TenorFlow+TensorRT while the second is TensorRT native. In this study, we have implemented the first option that involves TensorFlow integration with TensorRT on Jetson Xavier NX. However, the challenge in Jetson Xavier NX is its limited memory that does not provide enough space to convert the TensorFlow deep learning models into TensorRT format. To handle this issue, we cannot build a TensorRT execution engine on any other device to get the conversion results because it should be run on the GPU of the same device, on which inference will be performed. As a solution, we have created a swap file that allows Jetson Xavier NX to use more memory than its physically installed memory. For this, we built a memory swapfile on the SSD and leveraged the 24GB swap memory with 8 GB Jetson Xavier memory.

After that, we accomplished the optimization by following the steps shown in Figure 7.

First, we created a batched input to load test data and performed the batch processing on the images. Initially, we used a batch size of 8 images from 1000 images but the Jetson

Xavier NX ran out of memory at an early stage due to heavy computation in the TensorRT conversion process. To overcome this problem, we used a smaller batch size of 4. This batch processing promotes parallel processing and improves the throughput. Next, we start the implementation by loading our saved TensorFlow custom trained models for optimization and converting them to TensorRT format.

Finally, We obtain the prediction throughput and latency results of our base models and compare them with the throughput and latency of optimized TensorFlow+TensorRT models. For this, we converted and benchmarked TensorFlow models at FP32, FP16 and FP08 precisions. The results are discussed in section VI-B3.

C. VOICE MESSAGE ALERT SYSTEM

In this module, we have used PyAudio [118] library which is a Python binding for PortAudio. It is a cross platform library that allows to record and play audio at real-time. We call it to generate a voice alert message, when a detected person is not wearing a facae mask. We first recorded our voice message using PyAudio, ''You always have to wear a face mask.'' For this, we opened a stream with required audio parameters to setup the pyaudio. Stream to record audio is saved in voiceMessage.wav file. We execute this audio message using pyaudio.Stream.read() when a call to the voice message alert system is received.

V. EXPERIMENTS

A. DATASET

We have employed a Kaggle face mask detection dataset [119] that include 12,000 images which contains people wearing a mask and not wearing a mask both in indoor and outdoor environments. All mask images in this dataset have been collected from google search while face images without mask have been processed from CelebFace dataset [120]. We have made the dataset balanced for our work. It consists of 6000 face images with mask and 6000 face images without mask. The images in the dataset have different characteristics varying largely in poses, scales, illuminations, and occlusions. For example: In some images, the face occupies large portion in the image while in other images it occupies small portion; faces lying at different positions in the image i.e. left, right, center, or at the border of the image which are not visible completely and only a part of the face is depicted in the image; faces that are partially occluded by some objects and the faces that are not overlapped.

B. SOFTWARE SPECIFICATIONS

We have employed various tools in which TensorFlow [12], Keras [121] and Tnesorflow-TensorRT (TF-TRT) [122] are the main tools to implement and deploy the models. TensorFlow [12] is a DL framework that provides both high and low level APIs. It is written to provide support for Python, C++ and CUDA. It was developed by the Google Brains team to perform multiple machine learning tasks on different processing units such as CPUs, GPUs, and TPUs.

Keras [121] is a powerful and high-level neural network API that is written in python and it runs on the top of TensorFlow. It was developed by François Chollet. The architecture of TensorFlow is more flexible and complex than Keras. The TensorFlow framework and its high-level API Keras, provide the support of pre-trained models that can be used to solve a specific problem using Transfer Learning.

However, end-to-end inference of these models is slow on the edge devices. To overcome this challenge, TensorRT [122] was introduced by Nvidia to run faster inference. In this study, TensorRT integration with TensorFlow also known as TF-TRT has been implemented for inference optimization on Jetson Xavier NX.

C. HARDWARE SPECIFICATIONS

The main main components for this task are Jetson Xavier NX, Zed camera and USB speaker. Both camera and speaker have been connected with Jetson Xavier NX. The position of the robot's cameras on a mast is at human eye level that allows the robot to capture the image with the best possible angle which is suitable for optimal face mask recognition.

D. CASE STUDIES

COVID-19 pandemic has affected every aspect of human life in different environments including workplaces, educational institutes and hospitals. In order to prevent its spread and avoid the health risks, safety instructions have to be followed in both indoor and outdoor places by wearing a face mask as a precaution. In our case studies, we have conducted the experiments in both indoor and outdoor environments to perform face mask recognition with GuardBot.

1) MONITORING MODES

The GuardBot works in patrolling and non-patrolling modes to monitor the people for face mask recognition.

In patrolling mode, it performs multi-floor navigation in the indoor environment and eliminates the need for human workers to ensure the people are wearing the face masks while moving in the outdoor environment.

In non-patrolling mode, it stands stationary at a specific location to monitor the people wearing masks.

In both modes, it carries out a polite warning through a voice message if people are not wearing the face mask.

The workflow of experiments in both indoor and outdoor environments with two different modes is illustrated in following scenarios.

i) Scenario 1: Indoor Environment - Non-patrolling mode
In this scenario, we have placed the GuardBot at the entrance of the building. When a new visitor arrives at the entrance, it detects the human face to recognize the face mask by its entrance check system as illustrated in the Figure 8 and generates a voice message if that person is not wearing a mask.

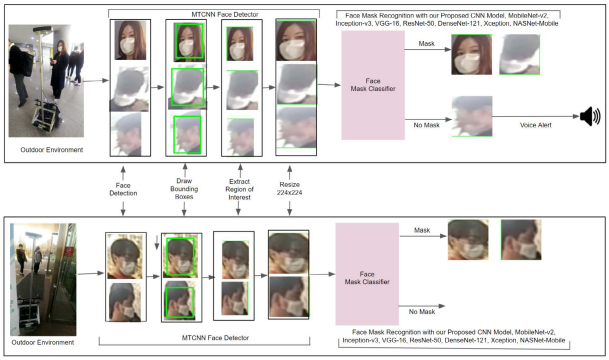

FIGURE 8. Indoor environment - GuardBot in non-patrolling mode & outdoor environment - GuadBot in patrolling mode.

Figure 8 shows that GuardBot's entrance check system is performing face detection and mask recognition in the indoor environment. Its workflow in indoor non-patrolling mode is as follows:

- Check the arrival of visitor(s) at the entrance
- Capture the face(s) for detection
- Perform the classification for face mask recognition
- Generate the polite voice message if the visitor entering in the building is not wearing face mask

ii) Scenario 2: Outdoor Environment - Patrolling Mode In this scenario, it patrols in the vicinity of the outdoors and detects the faces of the people to determine whether they wear the face mask or not as shown in Figure 8 and asks for wearing a mask if it finds a person without a face mask. Its pipeline in outdoor patrolling mode as shown in Figure 8 works as follows:

- Patrolling in open area
- Monitoring the people by detecting their faces
- Performing the face mask recognition
- Issuing voice warning message when people without mask are detected

VI. RESULTS AND DISCUSSION

In this section, we illustrate the experimental results in detail using evaluation matrices. To obtain the experimental results, we captured the live videos from the camera of GuardBot. We also acquired the experimental results from recorded video and static images. The results explanation with through discussion has been divided into two subsections. In first subsection VI-A, we describe the experimental results of face detection module together with the MTCNN optimization. In second subsection VI-B, we explain the results of face mask recognition module along with the models' optimization.

A. FACE DETECTION MODULE

The experimental results of the face detection module show the performance of four face detectors: Haar cascade [99], HOG [100], Caffe DNN [13] and MTCNN [98].

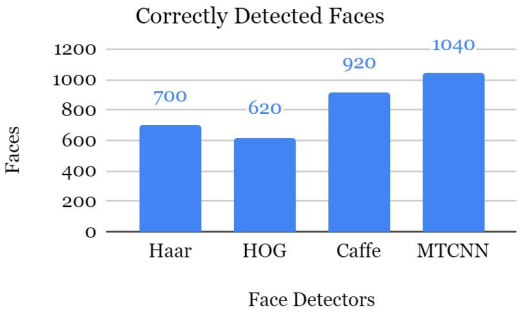

FIGURE 9. Correctly detected faces by four face detectors.

TABLE 3. Accuracy of face detectors.

Face Detectors	MTCNN	Caffe DNN	HOG	Haar Cascade
Accuracy	86.67%	76.96%	58.33%	51.48

The detection results of these four models have been compared based on their

- Accuracy
- FPS
- Time

1) ACCURACY

To determine the most accurate face detection model, we passed 300 images consisting of 1200 faces to each model. Experimental results show that

I) Face detection using Haar Cascade [99] is light-weight and properly detects frontal faces. However it is prone to false-positives and its accuracy is not high compared to other face detectors.

II) Face detection with HOG [100] is less accurate than Haar cascade [99] and computationally more expensive than Haar cascade. It fails to detect profile faces because it is not invariant to the changes in viewing angle.

III) Face detection with Caffe model [13] is more accurate than Haar cascade [99] and HOG [100]. It is capable of detecting the faces from varying angles. Since its backbone contains a shallow network with SSD [95], it is capable of running real-time.

IV) Face detection with MTCNN [98] is very accurate. Compared to all face detectors above, MTCNN has the best and the highest accuracy.

Figure 9 shows a graph with rectangular bars. The horizontal (x) axis represents the categories of face detectors; The vertical (y) axis represents a total number of 1200 faces. It illustrates that MTCNN has correctly detected the maximum number of 1040 faces with an 86.67% accuracy. Caffe DNN model correctly detected 920 faces with 76.96% accuracy, Haar cascade correcly detected 700 faces with 58.33% accuracy, while HOG correctly detected 620 faces and showed least 51.48% accuracy. Based on the results in Figure 9, the % accuracy of four models is shown in Table 3.

TABLE 4. Minimum and maximum number of frames processed for camera and recorded videos.

FPS Range		
Model	Camera	Recorded Video
Haar	2-9	8-14
HOG	1-2	3-3
Caffe	7-11	8-12
MTCNN	1	1-2

TABLE 5. Image processing time (in seconds) for camera and recorded video.

Processing Time (sec)		
Model	Camera	Recorded Video
Haar	4.26	1.91
HOG	39.8	23.66
Caffe	12.36	9.01
MTCNN	95.29	83.98

2) FPS: FRAME RANGE

Table 4 shows the frame range both in live camera and recorded video for each face detectors. It depicts a relatively better frame range for recorded video compared to the live camera on Jetson Xavier NX. Caffe DNN model performs face detection from minimum 7 FPS to maximum 11 FPS for live camera while 8-12 FPS for recorded video. It shows that MTCNN is very slow on Jetson Xavier NX despite its good accuracy because it requires high computation to run its 3-starge detection networks.

3) TIME

We have performed the face detection on 777 frames for both live camera and recorded video.

Table 5 shows that overall processing time of recorded video was less than the live camera video for each detector. It is the time that is taken by Jetson Xavier NX for each model to process images for face detection. Since, Jetson Xavier has limited memory, therefore, when we run MTCNN, it runs too slow and takes longer processing time. Because MTCNN consists of three CNNs, therefore, its processing speed is slow compared to other models and Jetson Xavier takes more time when we run MTCNN model. The results demonstrate that the Haar Cascade detector has the lowest processing time for both live camera and recorded video. MTCNN has the highest processing time compared to all other detectors on Jetson Xavier NX.

4) OPTIMIZATION

The accuracy of MTCNN is higher, however, its incoming frame processing speed is very slow and it takes more time. To handle this issue, we have used its open source implementation [116] that is optimized with TensorRT and adopted this implementation with some customization for our face recognition module. To evaluate the performance, we have used live camera and recorded videos of one minute. Table 6 reports that incoming frame processing has increased with descent FPS rate for both camera and videos at different resolutions. It also shows that processing time for recorded videos is less than the live camera videos and processing time increases for high resolution videos.

TABLE 6. Frames per Second with Optimized MTCNN at different resolutions.

Optimized MTCNN	FPS	Time (sec)	Resolution	Width	Height
Camera (Live Videos)	10	111	360	640	360
	11	147	480	854	480
	12	188	720	1280	720
Recorded Videos	12	80	360	640	360
	12	130	480	854	480
	10	176	720	1280	720

B. FACE MASK RECOGNITION MODULE

We have implemented eight face mask classification models for mask recognition. These models include our proposed CNN model, DenseNet121 [104], VGG16 [103], NASNetMobile [106], Inception-v3 [102], Xception [105], ResNet50 [91] and MobileNet-v2 [96].

The results in this section have been discussed in three parts. In the first part VI-B1, we start the discussion by analyzing the results with and without fine-tuning of the eight face mask classification models on the training dataset. Next, we examine the results of fine-tuned models based on training and validation dataset. In the second part VI-B2, the performance on testing dataset has been evaluated.

The classification results of eight models for face mask recognition have been evaluated based on their

- Accuracy
- Precision
- Recall
- F1-score
- Confusion Matrix

While in third part VI-B3, results before and after optimization have been discussed with the main focus on thier

- High throughput
- Low latency

1) ACCURACY: TRAINING AND VALIDATION DATASETS

The training loss in Figure 10 shows that all models converge faster after fine-tuning. A rapid reduction in loss from 5.792, 2.871 and 1.244 to 0.099, 0.343 and 0.094 can be seen in ResNet50, Xception and VGG16 models while a slight loss reduction of 0.032 can be seen in our model.

In addition, these models have gained better accuracy after fine-tuning as shown in Figure 11. From Figure 11, it can be seen that the accuracy of VGG16, NASNetMobile, ResNet50, DenseNet121 and our model has increased by 4.2, 4.1, 3.9, 2.7 and 1.6 respectively while the accuracy of Inception-v3 and MobileNet-v2 has improved by 1.6 after fine-tuning. However, a slight rise of 0.2 can be seen in case of Xception. The overall analysis on training set with fine-tuning shows that the DenseNet121 has the highest accuracy of 98%. After that NASNetMobile, ResNet50, VGG16, our CNN model

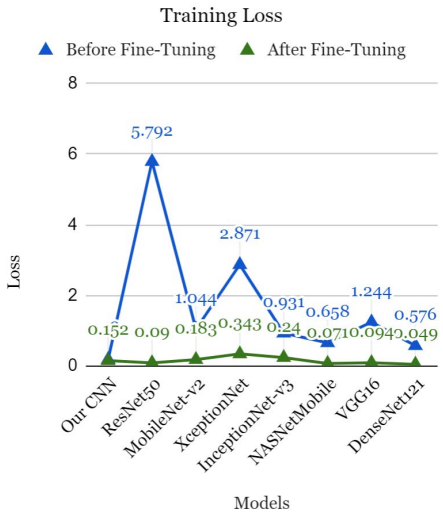

FIGURE 10. Training loss of models with and without fine-tuning.

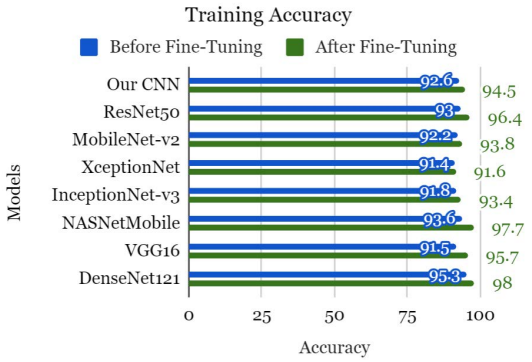

FIGURE 11. Training accuracy of models with and without fine-tuning.

have better performancce than MobileNet-v2 and Inception-v3. While Xception has the lowest training accuracy among all that is 91.6% to classify face mask for recognition task.

As discussed above, the fine-tuned models demonstrate better performance. Therefore, we have used fine-tuned models to perform the face mask recognition task. Table 7 gives an overview of fine-tuned models on the training and validation dataset based on their training loss and accuracy while Figure 12 shows the comparison of plots for the training and validation loss & accuracy of each model. It can be seen the validation loss being smaller than the training loss and slight increase in the validation accuracy compared to training accuracy for all the models. The smaller loss with an improved accuracy indicates better performance of the classifier at modeling the validation data.

2) ACCURACY: TEST DATA

In this subsection, the performance of fine-tuned models on the test dataset using confusion matrix, accuracy, precision, recall and F1- score has been evaluated.

TABLE 7. Accuracy vs loss for training and validation dataset.

		Training Dataset		Validation Dataset	
#	Models	Loss	Accuracy (%)	Loss	Accuracy (%)
1	Our CNN	0.152	94.5	0.068	95.9
2	ResNet50	0.090	96.4	0.011	96.5
3	MobileNet-v2	0.183	93.8	0.039	94.7
4	Xception	0.343	91.6	0.074	92.1
5	InceptionNet-v3	0.240	93.4	0.009	95.4
6	NASNetMobile	0.071	97.7	0.009	96.9
7	VGG16	0.094	95.7	0.009	98.4
8	DenseNet121	0.049	98.0	0.005	99.0

Figure 13, lists the confusion matrices of all classification models for the comparative analysis based on the frequency of predicted classes with the expected classes

It can be seen that DenseNet121 has relatively high accuracy compared to all other models for TP (348) where actual and predicted values are true and TN (2) where the actual and the predicted values both are false. In the meanwhile, it depicts low scores for FN (3) where actual value is true but the model predicted the value as false and FP (347) where the model predicted value is true, but the actual value is false. In contrast, MobileNet-v2 indicates low accuracy for TP (319) and TN (313) while high scores for FN (31) and FP (37). However, our proposed model has better classification results than Inception-v3, Xception and MobileNet-v2.

Table 8 shows an overview of the results of fine-tuned models that have been evaluated on the testing dataset using five metrics: accuracy, precision, recall and F1 score.

Figure 14, shows our CNN model is 2.08% better than Inception-v3, 2.99% than Xception and 3.99% more accurate than mobileNet-v3. The DenseNet121 is the most accurate compared to all other models while MobileNet-v2 depicts the lowest accurate model. NASNetMobile, VGG16 and RestNet50 have better performance than Inception-v3 and Xception models on the testing dataset.

Figure 15 shows that for Mask class, our model outperforms ResNet50, MobileNet-v2, Xception, InceptionNet-v3 and VGG16 on precision metric while it out performs MobileNet-v2 and Xception on recall metric and its F1-score is higher comparedto MobileNet-v2, Xception and VGG16. Figure 15 depicts that for No Mask class, its precision is higher than the Mobilenet-v2 and smiliar to Xception and InceptionNet-v3 while it outperforms all other models except NASNetMobile and DenseNet121 on recall metric and its peformance is better than MobileNet-v2, Xception and InceptionNet-v3 on F1-score. Figure 15 and Figure 16 show that DenseNet121 and NASNetMobile both outperform all other models regarding precision, recall and F1-score in both classes. It can be seen that the Inception-v3 model shows the same precision, recall and F1-score for "No Mask" class. However, it has 0.01 difference between, precision, F1-score and recall for "Mask" Class. The MobileNet-v2 has lowest precision and F1-score for both classes while its recall score is the lowest for Mask class. VGG16 and ResNet50 have the same recall and F1 score for "NO Mask" class. However,

FIGURE 12. Accuracy, loss, precision and recall curves for training and validation datasets.

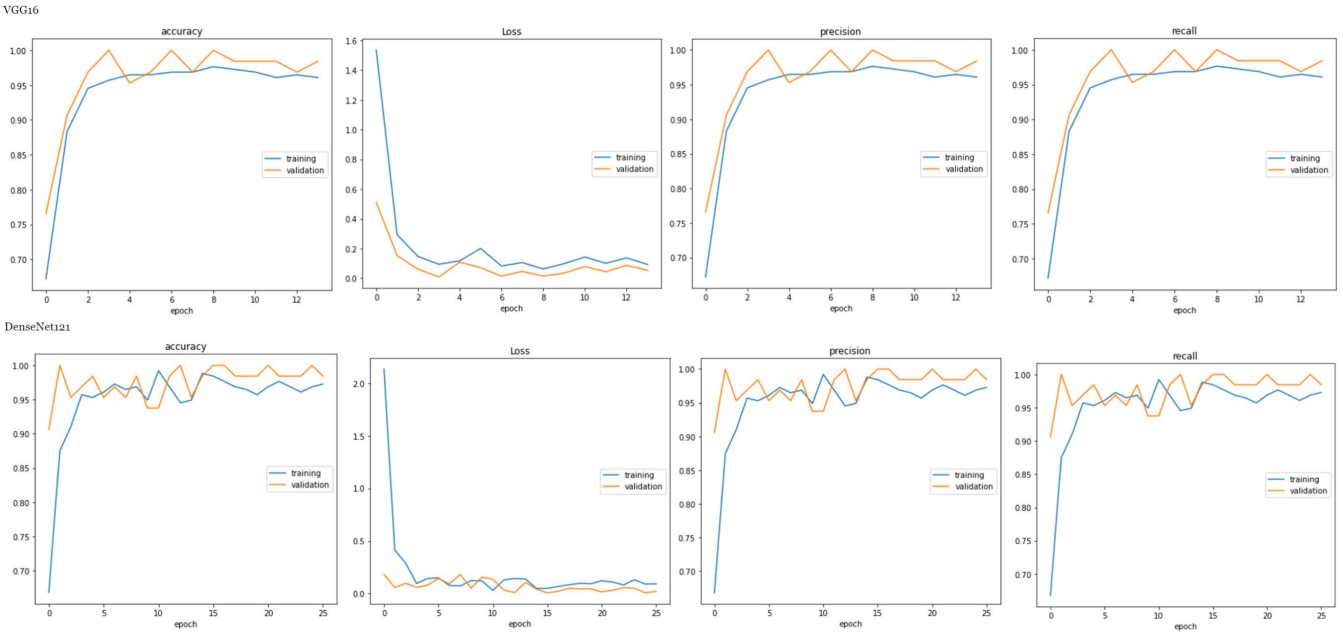

FIGURE 12. *(Continued.)* **Accuracy, loss, precision and recall curves for training and validation datasets.**

TABLE 8. **Accuracy, precision, recall and F1-score for test dataset.**

		Test Dataset						
		Accuracy	Precision		Recall		F1-Score	
#	Model		Mask	No Mask	Mask	No Mask	Mask	No Mask
1	Our CNN	94.28	0.97	0.92	0.92	0.97	0.94	0.94
2	ResNet50	95.57	0.95	0.96	0.96	0.95	0.96	0.96
3	MobileNet-v2	90.29	0.89	0.91	0.91	0.90	0.90	0.90
4	Xception	91.29	0.90	0.92	0.92	0.90	0.91	0.91
5	InceptionNet-v3	92.2	0.91	0.92	0.93	0.92	0.92	0.92
6	NASNetMobile	97.86	0.98	0.98	0.98	0.98	0.98	0.98
7	VGG16	96.0	0.95	0.97	0.97	0.95	0.96	0.96
8	DenseNet121	99.29	0.99	0.99	0.99	0.99	0.99	0.99

precision of VGG16 is .01 better than ResNet50 for this class. The Xception model lies between InceptionNet-v3 and MobileNet-v2 in terms of F1-score, its precision is .01 higher than the MobileNet-v2 while its recall is similiar to MobileNet-v2 for No Mask class. Its performance is between InceptionNet-v3 and MobileNet-v2 on precision, recall and F1-score metrics Mask class. On training, validation and test accuracy our model lies at 5th position as shown in Table 7 & Table 8 and performs better than MobilenNet-v2, Inception-v3 and XceptionNet.

3) OPTIMIZATION

In this section, we present the on-device inference performance of face mask classification models by focusing on the two main metrics: the latency & throughput and the trade-offs between them. The low latency and high throughput are the primary objectives when deploying the deep learning models on the edge devices and autonomous robotic systems. We have optimized our models with TensorRT which speeds up the inference by quantization for deployment in the production environment. We aim at improving the inference speed for our models with different optimization modes. For this, we have optimized our custom Keras-TensorFlow trained models with Nvidia's TensorRT at FP32, FP16, and INT8 precision, and observe performance effects along with inference throughput as shown in Table 9 and Figure 17.

In this section, we have used throughput and latency metrics to compare the performance of custom Keras-TensorFlow trained models for inference at the edge with Nvidia's TensorRT at FP32, FP16, and INT8 precision as shown in Table 9, Table 10 and Figure 17 & 18. The base models are our TensorFlow models without optimization while TF-TRT FP32, TF-TRT FP16 and TF-TRT INT8 are TensorFlow optimized models with TensorRT integration at different precision levels.

From the results shown in Table 9, it can be seen that our model outperforms all other models with the highest throughput among all base models (without optimization) and optimized FP32, FP16 and INT8 models which is followed by the MobileNet-v2 at second position when we

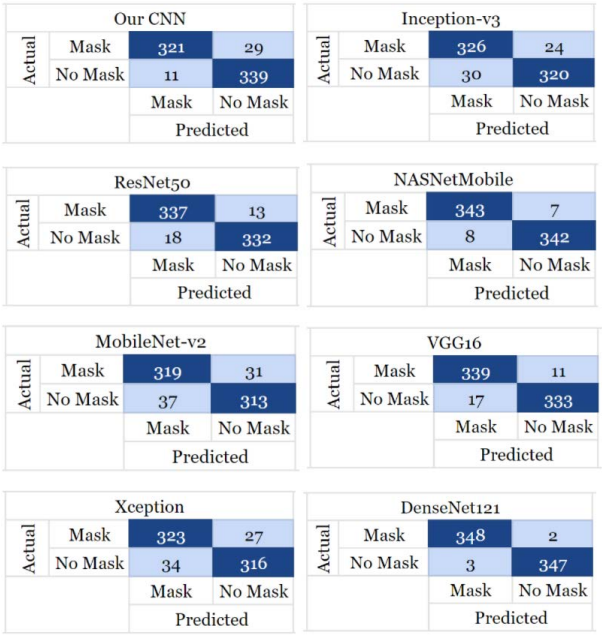

FIGURE 13. Confusion matrices of face mask classification models for face mask recognition.

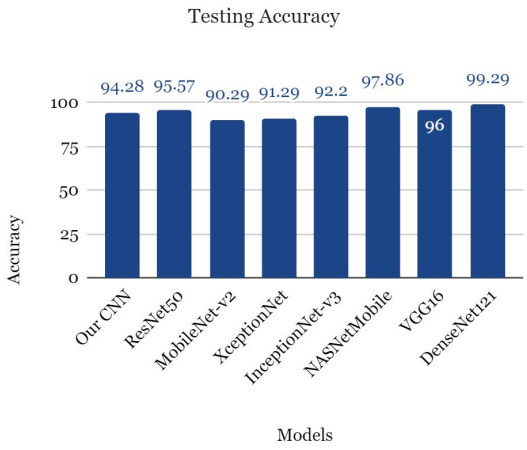

FIGURE 14. Accuracy on test dataset of face mask classification models for face mask recognition.

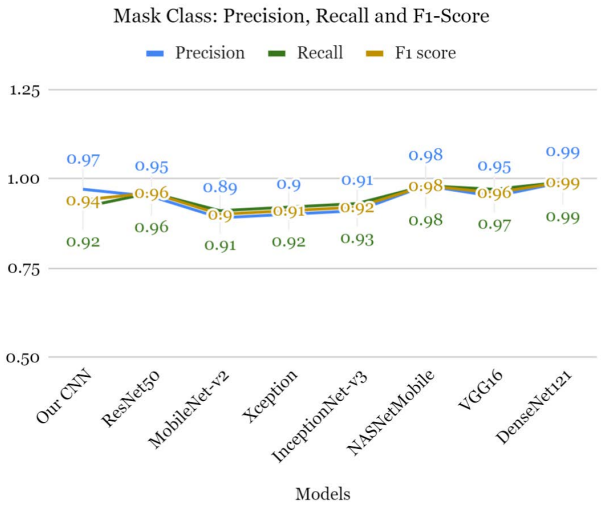

FIGURE 15. Precision, recall and F1-score for Mask class.

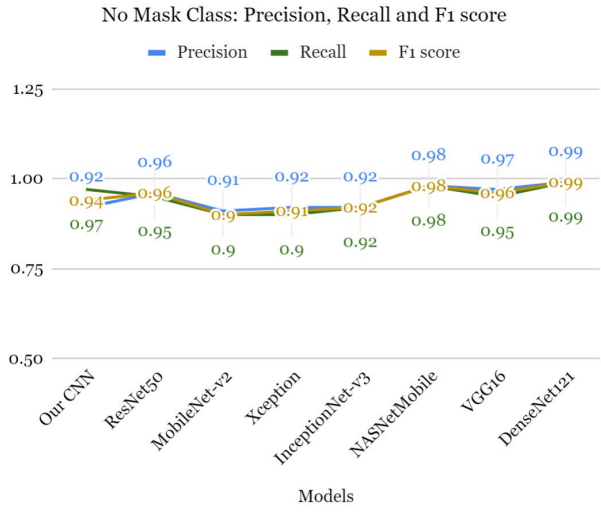

FIGURE 16. Precision, Recall and F1-score for No Mask class.

TABLE 9. Throughput of face mask classification models for face mask recognition.

#	Models	Models' Throughput Before and After Optimization			
		TF FP32 Base Model	TF TRT FP32	TF TRT FP16	TF TRT INT8
1	Our CNN	23	162	373	815
2	ResNet50	15	76	264	465
3	MobileNet-v2	20	148	351	799
4	Xception	14	55	176	339
5	InceptionNet-v3	15	106	339	513
6	NASNetMobile	15	114	173	234
7	VGG16	13	27	110	265
8	DenseNet121	16	79	183	227

run the inference on Jetson Xavier NX. For base models, the throughput of DenseNet121 is higher than the rest of five models, in which Xception and VGG16 have the least throughput of 14 and 13 images per second while NasNet-Mobile, Incpetion-v3 and ResNet have similar throughput of 15 images per second. For TF-TRT optimized models at FP32 precision level, the NASNetMobile followed by Inception-v3 and DensNet121 have higher throughput compared to the ResNet50, Xception and VGG16 models. For TF-TRT FP16 precision level, the Inception-v3, ResNet50 and DensNet121 have better throughput than the Xception, NASNet and VGG16. For TF-TRT INT8 precision level, Inception-v3, ResNet50 and Xception have better throughput compared to VGG16, NASNetMobile and DenseNet121.

Table 10 shows the results of our custom trained models based on latency metric for inference at edge deployment on Jetson Xavier NX. It can be that our model has the

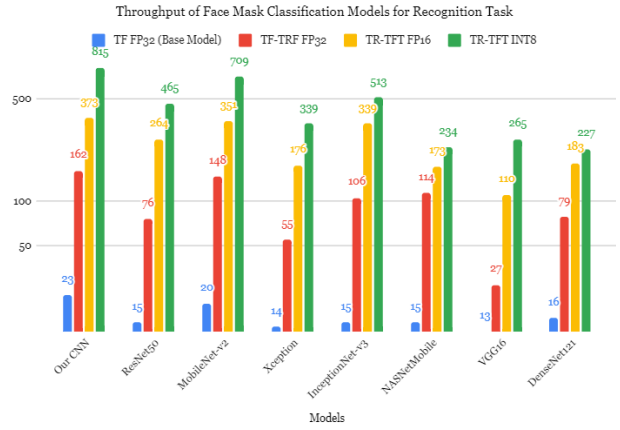

FIGURE 17. Models' throughput before and after optimization at different precision levels.

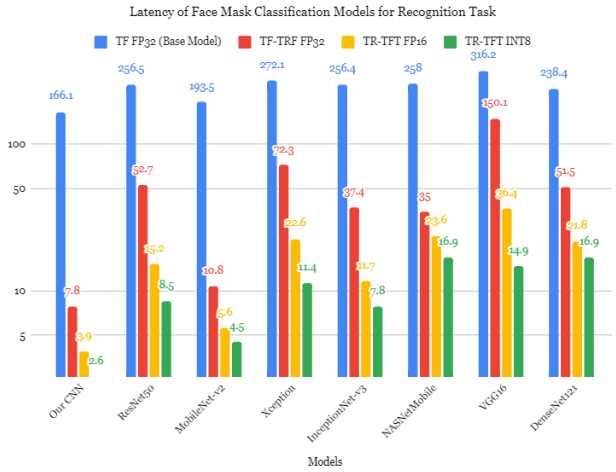

FIGURE 18. Models' latency before and after optimization at different precision levels.

TABLE 10. Latency of face mask classification models for face mask recognition.

#	Models	Models' Latency Before and After Optimization			
		TF FP32 Base Model	TF TRT FP32	TF TRT FP16	TF TRT Int8
1	Our CNN	166.1	7.8	3.9	2.6
2	ResNet50	256.5	52.7	15.2	8.5
3	MobileNet-v2	193.5	10.8	5.6	4.5
4	Xception	272.1	72.3	22.6	11.4
5	InceptionNet-v3	256.4	37.4	11.7	7.8
6	NASNetMobile	258.0	35.0	23.6	16.9
7	VGG16	316.2	150.1	36.4	14.9
8	DenseNet121	238.4	51.5	21.8	16.9

lowest latency rate compared to all other models. After that, MobileNet-v2 has the maximum reduction in latency of all the models. For base models, DenseNet121 has lower latency than Xception and Inception-v3 models while Inception-v3 and ResNet18 have almost similar latency which is less than NASNetMobile and VGG16 For TF-TRT FP32 optimization, NASNetMobile has lower latency than Inception-v3 and DenseNet121 while VGG16 has higher latency than Xception and ResNet50. For INT8 optimization, Inception-v3 has lower latency compared to ResNet50, Xception and VGG16 while DenseNet121 and NASNetMobile have almost similar latency.

In addition, it is observed that compared to base models when we run the TensorFlow models with TensorRT intergration for optimization to speed up the inference

- Using TF-TRT FP32 optimization improves the throughput of these models by 107.69% (27 vs 13) to 660.00% (114 vs 15) which indicates a 2× to 7× performance increase.
  The throughput of our model has increased 7.0 times compared to its base model. After that, NASNetMobbile model shows the maximum increase in performance and it is 0.6 times faster than our model while VGG16 shows the minimum increase compared to other models and it is 5× slower than our model.
  At the same time FP32 optimization reduces models' latency in the range of 52%∼95%. VGG16 has the minimum reduction of 52% while our model presents the maximum reduction of 95% (from 166.1ms to 7.8ms) in latency among all models which is 21 times lesser than its base model.
- Using FP16 optimization the inference speed increases by 746.15% (110 vs 130) ∼ 2160.00% (339 vs 15) which shows the performance of these models is 8 to 22 times higher than the base models. The performance of our model is 16 times faster than the base model while the Inception-v3 with increased throughput is 6 times faster and VGG16 is 7.8 times slower compared to our model. Meanwhile the maximum decrease in latency has been encountered in our model by 97% (from 166.1 ms to 3.9 ms) while the minimum decrease has been experienced in the VGG16 model by 88% (from 316.2 ms to 36.4 ms). Our model has a 42× decrease in latency while VGG16 has 8× decrease compared to their base models' latency.
- Using INT8, we examine a speedup of 1318.75% (227 vs 16) to 3895.00% (799 vs 20) which exhibits 14 to 39 times faster performance. The inference speed of our model rises 35 times faster than the base model. MobileNet-v2 has the highest performance increase that is 4.5 times better than our model while DenseNet121 is 21.25 times slower compared to our model. Simultaneously, Similarly, the decline in latency has been observed by 92.9% 98.4%. Our model has the lowest latency with 98% decrease which makes a 63× reduction in latency compared to its base model. While DenseNet121 has the highest latency at INt8 precision among all models that is 92.9%. Its latency is 14× lower than the base model and 5.5% higher than our model at INT8.
  To summarize, our model has the highest throughput and lowest latency because the architecture of our model is tiny which does not require large amount of computational memory compared to other models. After that the throughput and latency of MobileNet-v2 is better than other models because MobileNet-v2 also has light

weight architecture which does not required huge computation compared to the XceptionNet, NASNetMobile, DenseNet121, VGG16, IncpetionNet-v3 and ResNet50 models.

VII. CONCLUSION AND FUTURE WORK

This paper presented a two-stage face mask recognition framework that was optimized to run real-time inference on the GuartBot using Nvidia Jetson NX. Our DL-based framework was composed of three main modules. It processed every frame for face detection using MTCNN in the first module and performed the face mask recognition based on the classification using our proposed CNN model and seven transfer learning-based custom models in the second module while optimization was performed in the third module using TensorRT integration with TensorFlow. It set a polite voice message to request wear a mask if a person without face mask was identified. The real-world experiments were conducted in the indoor and outdoor environments with the GuardBot in patrolling and non-patrolling modes and the results were discussed in detail. We first discussed the results on the training and validation datasets based on accuracy and loss curves of different models. After that, we evaluated the performance of all the models on the test dataset by presenting the classification report using the confusion matrix and analyzed their predictions based on accuracy, recall, precision and F1-score metrics. Finally, all the models were optimized with TF-TRT FP32, FP16 and INT8 precision with the main aim of reducing the latency and improving the throughput. Experimental results demonstrated the effectiveness of our proposed CNN model to classify the face masks for the recognition task with an accuracy higher than MobileNet-v2, Xception and InceptionNet-v3 while efficient inference with the highest throughput and lowest latency compared to all other models after optimization.

In future, we intend to extend our framework for detection-based human tracking problem and examine the impact of our framework optimizations on the detection and tracking accuracy across different edge devices.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, Dec. 2015.

[2] Y. Chen, Q. Feng, and W. Shi, "An industrial robot system based on edge computing: An early experience," in *Proc. USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, 2018, pp. 1–6.

[3] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, and Q. Zhang, "Edge computing in IoT-based manufacturing," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 103–109, Sep. 2018.

[4] V. Combs. (Jun. 4, 2022). *Will Edge Computing Become the New Cloud in 2021?* [Online]. Available: https://www.techrepublic.com/article/willedge-computing-become-the-new-cloud-in-2021/

[5] A. Weil. (Mar. 3, 2022). *Can Edge Computing Exist Without the Edge? Part 2: Edge Computing*. [Online]. Available: https://www.akamai.com/blog/edge/can-edge-computingexist-without-the-edge-part-2-edge-computing

[6] IDC. (Mar. 4, 2022). *Worldwide Spending on Edge Computing Will Reach $250 Billion in 2024, According to a New IDC Spending Guide*. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS46878020

[7] Omdia. (Mar. 4, 2022). *Worldwide Spending on Edge Computing Will Reach $250 Billion in 2024, According to a New IDC Spending Guide*. [Online]. Available: https://omdia.tech.informa.com/topic-pages/artificial-intelligence

[8] Z. Tsai. (Mar. 5, 2022). *The Emerging Role of AI in Edge Computing*. [Online]. Available: https://www.adlinktech.com/en/emergingrole-of-ai-in-edge-computing

[9] R. Mahmud and A. N. Toosi, "Con-Pi: A distributed container-based edge and fog computing framework," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4125–4138, Mar. 2021.

[10] Z. Tsai. *The Emerging Role of AI in Edge Computing*. [Online]. Available: https://nvidianews.nvidia.com/news/nvidia-sets-path-for-future-of-edge-%ai-and-autonomous-machines-with-new-jetson-agx-orin-robotics-computer

[11] Y. Wang, "Towards ultra-efficient DNN inference acceleration on edge devices for wellbeing applications," in *Proc. Deep Learn. Wellbeing Appl. Leveraging Mobile Devices Edge Comput.*, Jun. 2020, p. 17.

[12] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, and L. Antiga, "Pytorch: An imperative style, highperformance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[15] J. Redmon. (Apr. 6, 2022). *DarkNet: Open Source Neural Networks in C*. [Online]. Available: https://pjreddie.com/darknet/

[16] Paddle. (Apr. 6, 2022). *Paddle Deep Learning*. [Online]. Available: https://github.com/PaddlePaddle/Paddle

[17] G. Verma, Y. Gupta, A. M. Malik, and B. Chapman, "Performance evaluation of deep learning compilers for edge inference," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, Jun. 2021, pp. 858–865.

[18] J. Roesch, S. Lyubomirsky, M. Kirisame, L. Weber, J. Pollock, L. Vega, Z. Jiang, T. Chen, T. Moreau, and Z. Tatlock, "Relay: A high-level compiler for deep learning," 2019, *arXiv:1904.08368*.

[19] A. S. J. Pool and J. Rodge. *Accelerating Inference With Sparsity Using the NVIDIA Ampere Architecture and NVIDIA Tensorrt*. [Online]. Available: https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-%using-ampere-and-tensorrt/

[20] S. Li, "Tensorflow lite: On-device machine learning framework," *J. Comput. Res. Develop.*, vol. 57, no. 9, p. 1839, 2020.

[21] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, and L. Ceze, "TVM: An automated end-to-end optimizing compiler for deep learning," in *Proc. 13th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2018, pp. 578–594.

[22] T. Kumar, "An evidence review of face masks against covid-19," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 9, no. 12, pp. 919–923, Dec. 2021.

[23] Q. Wang and C. Yu, "The role of masks and respirator protection against SARS-CoV-2," *Infection Control Hospital Epidemiol.*, vol. 41, no. 6, pp. 746–747, Jun. 2020.

[24] POLARIS. (Feb. 25, 2022). *Service Robotics Market Size to Reach USD 54.4 Billion by 2026*. [Online]. Available: https://www.polarismarketresearch.com/press-releases/global-service-robotics-market

[25] F. 24. (Feb. 25, 2022). *Robot Reminds Japan Shoppers to Wear Masks*. [Online]. Available: https://www.france24.com/en/live-news/20201119-robot-reminds-japan-shoppers-to-wear-masks

[26] S. B. Robotics. (Mar. 25, 2022). *Protect Your Customers & Employees With Pepper's Mask Detection Capabilities!* [Online]. Available: https://www.softbankrobotics.com/emea/fr/pepper-mask-detection-capabilities

[27] F. Niroui, K. Zhang, Z. Kashino, and G. Nejat, "Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 610–617, Apr. 2019.

[28] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Comput. Intell. Neurosci.*, vol. 2018, pp. 1–13, Feb. 2018.

[29] S. Manzoor, E.-J. Kim, G.-G. In, and T.-Y. Kuc, "Performance evaluation of YOLOv3 and YOLOv4 detectors on elevator button dataset for mobile robot," in *Proc. 21st Int. Conf. Control, Autom. Syst. (ICCAS)*, Oct. 2021, pp. 890–893.

[30] K.-H. L. Minh, K.-H. Le, and Q. Le-Trung, "DLASE: A light-weight framework supporting deep learning for edge devices," in *Proc. 4th Int. Conf. Recent Adv. Signal Process., Telecommun. Comput. (SigTelCom)*, Aug. 2020, pp. 103–108.

[31] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 1421–1429.

[32] P. Liu, B. Qi, and S. Banerjee, "EdgeEye: An edge service framework for real-time intelligent video analytics," in *Proc. 1st Int. Workshop Edge Syst., Analytics Netw.*, Jun. 2018, pp. 1–6.

[33] S. Manzoor, S.-H. Joo, E.-J. Kim, S.-H. Bae, G.-G. In, J.-W. Pyo, and T.-Y. Kuc, "3D recognition based on sensor modalities for robotic systems: A survey," *Sensors*, vol. 21, no. 21, p. 7120, Oct. 2021.

[34] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, Aug. 2019, pp. 1–16.

[35] A. Koubaa, A. Ammar, M. Alahdab, A. Kanhouch, and A. T. Azar, "DeepBrain: Experimental evaluation of cloud-based computation offloading and edge computing in the Internet-of-Drones for deep learning applications," *Sensors*, vol. 20, no. 18, p. 5240, Sep. 2020.

[36] B. Taylor, V. S. Marco, W. Wolff, Y. Elkhatib, and Z. Wang, "Adaptive deep learning model selection on embedded systems," *ACM SIGPLAN Notices*, vol. 53, no. 6, pp. 31–43, Dec. 2018.

[37] M. Alzantot, Y. Wang, Z. Ren, and M. B. Srivastava, "RSTensorFlow: GPU enabled TensorFlow for deep learning on commodity Android devices," in *Proc. 1st Int. Workshop Deep Learn. Mobile Syst. Appl. (EMDL)*, 2017, pp. 7–12.

[38] U. Drolia, K. Guo, and P. Narasimhan, "Precog: Prefetching for image recognition applications at the edge," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, 2017, pp. 1–13.

[39] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2016, pp. 1–12.

[40] Y. Huang, X. Ma, X. Fan, J. Liu, and W. Gong, "When deep learning meets edge computing," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–2.

[41] M. Xu, F. Qian, M. Zhu, F. Huang, S. Pushp, and X. Liu, "DeepWear: Adaptive local offloading for on-wearable deep learning," *IEEE Trans. Mobile Comput.*, vol. 19, no. 2, pp. 314–330, Feb. 2020.

[42] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, Aug. 2018.

[43] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE Access*, vol. 7, pp. 19143–19165, 2019.

[44] X. Wang, M. Hersche, B. Tomekce, B. Kaya, M. Magno, and L. Benini, "An accurate EEGNet-based motor-imagery brain–computer interface for low-power edge computing," in *Proc. IEEE Int. Symp. Med. Meas. Appl. (MeMeA)*, Jun. 2020, pp. 1–6.

[45] J. Ker, L. Wang, J. Rao, and T. Lim, "Deep learning applications in medical image analysis," *IEEE Access*, vol. 6, pp. 9375–9389, 2018.

[46] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Proc. 40th Int. Conf. Softw. Eng.*, May 2018, pp. 303–314.

[47] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[48] S. Manzoor, S.-H. Joo, and T.-Y. Kuc, "Comparison of object recognition approaches using traditional machine vision and modern deep learning techniques for mobile robot," in *Proc. 19th Int. Conf. Control, Autom. Syst. (ICCAS)*, Oct. 2019, pp. 1316–1321.

[49] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," 2020, *arXiv:2007.05558*.

[50] M. N. Birje, P. S. Challagidad, R. Goudar, and M. T. Tapale, "Cloud computing review: Concepts, technology, challenges and security," *Int. J. Cloud Comput.*, vol. 6, no. 1, pp. 32–57, 2017.

[51] R. Sharp, "Latency in cloud-based interactive streaming content," *Bell Labs Tech. J.*, vol. 17, no. 2, pp. 67–80, Sep. 2012.

[52] H. Takabi, J. B. D. Joshi, and G.-J. Ahn, "Security and privacy challenges in cloud computing environments," *IEEE Security Privacy*, vol. 8, no. 6, pp. 24–31, Nov./Dec. 2010.

[53] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[54] Thales. (Mar. 28, 2022). *2021 Thales Cloud Security Study*. [Online]. Available: https://www.thalesgroup.com/en/poland/press_release/majority-businesses-protect-theirsensitive-data-cloud-finds-thales

[55] N. Subramanian and A. Jeyaraj, "Recent security challenges in cloud computing," *Comput. Electr. Eng.*, vol. 71, pp. 28–42, Oct. 2018.

[56] N. Hassan, K.-L. A. Yau, and C. Wu, "Edge computing in 5G: A review," *IEEE Access*, vol. 7, pp. 127276–127289, 2019.

[57] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of Things: Vision, applications and research challenges," *Ad Hoc Netw.*, vol. 10, no. 7, pp. 1497–1516, 2012.

[58] H.-C. Hsieh, J.-L. Chen, and A. Benslimane, "5G virtualized multi-access edge computing platform for IoT applications," *J. Netw. Comput. Appl.*, vol. 115, pp. 94–102, Aug. 2018.

[59] S. Wan, Z. Gu, and Q. Ni, "Cognitive computing and wireless communications on the edge for healthcare service robots," *Comput. Commun.*, vol. 149, pp. 99–106, Jan. 2020.

[60] C. Yvanoff-Frenchin, V. Ramos, T. Belabed, and C. Valderrama, "Edge computing robot interface for automatic elderly mental health care based on voice," *Electronics*, vol. 9, no. 3, p. 419, Feb. 2020.

[61] J. Zhu, Y. Chen, M. Zhang, Q. Chen, Y. Guo, H. Min, and Z. Chen, "An edge computing platform of guide-dog robot for visually impaired," in *Proc. IEEE 14th Int. Symp. Auto. Decentralized Syst. (ISADS)*, Apr. 2019, pp. 1–7.

[62] M. Zhang, F. Zhang, N. D. Lane, Y. Shu, X. Zeng, B. Fang, S. Yan, and H. Xu, "Deep learning in the era of edge computing: Challenges and opportunities," in *Fog Computing: Theory and Practice*. Wiley, 2020, pp. 67–78.

[63] J. Redmon. (Mar. 29, 2022). *DarkNet: Open Source Neural Networks in C*. [Online]. Available: https://pjreddie.com/darknet/

[64] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[65] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," 2014, *arXiv:1412.7024*.

[66] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," 2014, *arXiv:1412.6115*.

[67] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Deep Learn. Unsupervised Feature Learn. Workshop, NIPS Workshop*, Spain, vol. 1, 2011.

[68] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, and L. Hornof, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[69] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

[70] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 548–560, 2017.

[71] J. Lin, Y. Rao, J. Lu, and J. Zhou, "Runtime neural pruning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[72] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 1–9.

[73] H. Vanholder, "Efficient inference with tensorrt," in *Proc. GPU Technol. Conf.*, vol. 1, 2016, p. 2.

[74] A. Koubaa, A. Ammar, A. Kanhouch, and Y. AlHabashi, "Cloud versus edge deployment strategies of real-time face recognition inference," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 1, pp. 143–160, Jan. 2021.

[75] E. L. Schiffrin, J. M. Flack, S. Ito, P. Muntner, and R. C. Webb, "Hypertension and COVID-19," *Amer. J. Hypertension*, vol. 33, no. 5, pp. 373–374, 2020.

[76] CDC. *How COVID-19 Spreads*. [Online]. Available: https://www.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/how-covid-spreads.html

[77] *Risk Assessment and Management of Exposure of Health Care Workers in the Context of COVID-19: Interim Guidance*," World Health Org., Geneva, Switzerland, 2020.
[78] *Mask use in the Context of COVID-19: Interim Guidance*, World Health Org., Geneva, Switzerland, 2020.
[79] M. R. Prusty, V. Tripathi, and A. Dubey, "A novel data augmentation approach for mask detection using deep transfer learning," *Intelligence-Based Med.*, vol. 5, 2021, Art. no. 100037.
[80] S. Roy, A. Mitra, and S. K. Setua, "Color & grayscale image representation using multivector," in *Proc. 3rd Int. Conf. Comput., Commun., Control Inf. Technol. (CIT)*, Feb. 2015, pp. 1–6.
[81] P. Singhal, A. Verma, and A. Garg, "A study in finding effectiveness of Gaussian blur filter over bilateral filter in natural scenes for graph based image segmentation," in *Proc. 4th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, Jan. 2017, pp. 1–6.
[82] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.
[83] Z. Wang, P. Wang, P. C. Louis, L. E. Wheless, and Y. Huo, "WearMask: Fast in-browser face mask detection with serverless edge computing for COVID-19," 2021, *arXiv:2101.00784*.
[84] Nihui. (Feb. 25, 2022). *Neural Network Inference Computing (NCNN) Framework*. [Online]. Available: https://github.com/Tencent/ncnn
[85] S. Ge, J. Li, Q. Ye, and Z. Luo, "Detecting masked faces in the wild with LLE-CNNs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 2682–2690.
[86] Z. Wang, G. Wang, B. Huang, Z. Xiong, Q. Hong, H. Wu, P. Yi, K. Jiang, N. Wang, Y. Pei, H. Chen, Y. Miao, Z. Huang, and J. Liang, "Masked face recognition dataset and application," 2020, *arXiv:2003.09093*.
[87] H. in the Loop. (Feb. 26, 2022). *Medical Mask Dataset*. [Online]. Available: https://humansintheloop.org/medical-mask-dataset
[88] Prajnasb. (Feb. 25, 2022). *Observations*. [Online]. Available: https://github.com/prajnasb/observations
[89] S. Yang, P. Luo, C. C. Loy, and X. Tang, "WIDER FACE: A face detection benchmark," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 5525–5533.
[90] M. Loey, G. Manogaran, M. H. N. Taha, and N. E. M. Khalifa, "A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the COVID-19 pandemic," *Measurement*, vol. 167, Jan. 2021, Art. no. 108288.
[91] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.
[92] M. Kawulok, E. Celebi, and B. Smolka, *Advances in Face Detection and Facial Image Analysis*. Springer, 2016. [Online]. Available: https://books.google.co.kr/books?id=rsntCwAAQBAJ
[93] M. D. Putro, D.-L. Nguyen, and K.-H. Jo, "Real-time multi-view face mask detector on edge device for supporting service robots in the COVID-19 pandemic," in *Proc. Asian Conf. Intell. Inf. Database Syst.* Thailand: Springer, 2021, pp. 507–517.
[94] P. Nagrath, R. Jain, A. Madan, R. Arora, P. Kataria, and J. Hemanth, "SSDMNV2: A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2," *Sustain. Cities Soc.*, vol. 66, Mar. 2021, Art. no. 102692.
[95] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. Eur. Conf. Comput. Vis.* The Netherlands: Springer, 2016, pp. 21–37.
[96] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[97] S. V. Militante and N. V. Dionisio, "Real-time facemask recognition with alarm system using deep learning," in *Proc. 11th IEEE Control Syst. Graduate Res. Colloq. (ICSGRC)*, Aug. 2020, pp. 106–110.
[98] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Process. Lett.*, vol. 23, no. 10, pp. 1499–1503, Oct. 2016.
[99] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Dec. 2001, pp. 1–11.
[100] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, no. 1, Jun. 2005, pp. 886–893.
[101] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1995.
[102] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Dec. 2016, pp. 2818–2826.
[103] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[104] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.
[105] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 1251–1258.
[106] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.
[107] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on CIFAR-10," *Unpublished Manuscript*, vol. 40, no. 7, pp. 1–9, 2010.
[108] M. Sandler and A. Howard. (Apr. 3, 2018). *Google AI Blog*. Google Research. [Online]. Available: https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html
[109] F. Saxen, P. Werner, S. Handrich, E. Othman, L. Dinges, and A. Al-Hamadi, "Face attribute detection with MobileNetV2 and NasNet-mobile," in *Proc. 11th Int. Symp. Image Signal Process. Anal. (ISPA)*, Sep. 2019, pp. 176–180.
[110] (Jan. 1, 2022). *Supported Operators*. NVIDIA Accelerating Inference in TF-TRT User Guide. [Online]. Available: https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index%.html#supported-ops
[111] I. Sobel and G. Feldman, "A 3×3 isotropic gradient operator for image processing," in *Pattern Classification and Scene Analysis*, Jan. 1973, pp. 271–272.
[112] IPAZC. (Mar. 7, 2022). *MTCNN*. [Online]. Available: https://github.com/ipazc/mtcnn
[113] Jbrownlee. (Mar. 7, 2022). *MTCNN*. [Online]. Available: https://github.com/jbrownlee/mtcnn
[114] Linxiaohui. (Mar. 7, 2022). *MTCNN*. [Online]. Available: https://github.com/linxiaohui/mtcnn-opencv
[115] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 815–823.
[116] J. Jung. *Optimizing Tensorrt MTCNN*. JK Jung's Blog. (Oct. 5, 2019). [Online]. Available: https://jkjung-avt.github.io/optimize-mtcnn/
[117] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[118] (Mar. 3, 2022). *PyAudio*. [Online]. Available: http://people.csail.mit.edu/hubert/pyaudio/
[119] A. Jangra. (Mar. 7, 2022). *Face Mask Detection 12k Images Dataset*. [Online]. Available: https://www.kaggle.com/datasets/ashishjangra27/face-mask-12k-images-dataset
[120] Z. Liu, P. Luo, X. Wang, and X. Tang. (Mar. 7, 2022). *Large-Scale Celebfaces Attributes (Celeba) Dataset*. [Online]. Available: http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html
[121] (2020). K. Team. *Keras Documentation: Keras Applications*. Inglés. [Online]. Available: https://keras.io/api/applications
[122] S. S. J. Park, S. Rella, and H. Abbasian. (Jul. 20, 2021). *Speeding up Deep Learning Inference Using NVIDIA Tensorrt (Updated)*. Technical Blog. [Online]. Available: https://developer.nvidia.com/blog/speeding-up-deep-learning-inference-using-tensorrt-updated/

SUMAIRA MANZOOR received the M.S. degree in computer science from COMSATS University, Pakistan, in 2016. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Sungkyunkwan University, South Korea. She worked as a Researcher with the Intelligent System Research Institute, Sungkyunkwan University, in 2017, and joined the Control and Robotics Laboratory, in 2018. Since 2021, she has been working with the Creative Algorithms and Sensor Evolution Laboratory. Her major research interests include human–robot interaction, cognitive vision for robotics, and autonomous driving vehicles.

EUN-JIN KIM received the B.S. degree from the Division of Electronic Engineering, College of ICT, Dongeui University, in 2017. He is currently pursuing the Ph.D. degree with the School of Electrical and Electronics Engineering, Sungkyunkwan University, Suwon, South Korea. He has been with the School of Electrical and Electronics Engineering, Sungkyunkwan University. His research interests include coverage path planning and robot design.

SUNG-HYEON JOO received the B.S. degree from the School of Electrical and Electronics Engineering, Sungkyunkwan University, Suwon, South Korea, in 2017, where he is currently pursuing the Ph.D. degree. He has been with the School of Electrical and Electronics Engineering, Sungkyunkwan University. His research interests include mobile robot navigation and semantic simultaneous localization and mapping.

SANG-HYEON BAE received the B.S. degree from the School of Electrical and Electronics Engineering, Sungkyunkwan University, Suwon, South Korea, in 2017, where he is currently pursuing the Ph.D. degree. He has been with the School of Electrical and Electronics Engineering, Sungkyunkwan University. His research interests include mobile robot control and planning.

GUN-GYO IN received the B.S. degree from the Division of Electronic Engineering, Korea Polytechnic University, in 2017. He is currently pursuing the Ph.D. degree with the School of Electrical and Electronics Engineering, Sungkyunkwan University, Suwon, South Korea. He has been with the School of Electrical and Electronics Engineering, Sungkyunkwan University. His research interests include multiple robot planning and task allocation.

KYEONG-JIN JOO received the B.S. degree from the Department of Electronic Engineering, Dong-A University, South Korea, in February 2021. He is currently pursuing the integrated Ph.D. degree with the Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, South Korea. Since March 2021, he has been with the Department of Electrical and Computer Engineering, Sungkyunkwan University. His research interests include intelligent robotics, computer vision, SLAM, and navigation for the robot.

JUN-HYEON CHOI received the B.S. degree from the School of Electronic Engineering, Dong-A University, in 2021. He is currently pursuing the Ph.D. degree with the School of Electrical and Electronics Engineering, Sunkyunkwan University, Suwon, South Korea. He has been with the School of Electrical and Electronics Engineering, Sunkyunkwan University. His major research interests include cognitive vision and SLAM for mobile robot.

TAE-YONG KUC received the B.S. degree in control and instrumentation engineering from Seoul National University, South Korea, in 1988, and the M.S. and Ph.D. degrees from the Pohang University of Science and Technology, South Korea, in 1990 and 1993, respectively. From April 1993 to August 1993, he worked as a Chief Research Engineer at the Precision Machinery Institute of Samsung Aerospace Company. From September 1993 to February 1995, he was a Senior Lecturer with the Department of Electrical Engineering, Mokpo National University, South Korea. Since March 1995, he has been with the School of Electrical and Electronics Engineering, Sungkyunkwan University, Suwon, South Korea, where he is currently a Professor. His research interests include intelligent robotics, adaptive and learning control, and visual sensor processing for computer-aided control systems.